AFRL-IF-RS-TR-2004-179
Final Technical Report
June 2004

EMERGEANT: A TOOLKIT TO CREATE RUN-TIME AUTONOMOUS NEGOTIATING TEAMS (ANT) GENERATORS, AGGREGATORS AND SYNTHESIZERS

Kestrel Institute

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J123

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-179 has been reviewed and is approved for publication

APPROVED: /s/

ROBERT J. PARAGI
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 2004	Final Jan 00 – Dec 03

4. TITLE AND SUBTITLE
EMERGEANT: A TOOLKIT TO CREATE RUN-TIME AUTONOMOUS NEGOTIATING TEAMS (ANT) GENERATORS, AGGREGATORS AND SYNTHESIZERS

5. FUNDING NUMBERS
C - F30602-00-C-0014
PE - 62301E
PR - J123
TA - 01
WU - 01

6. AUTHOR(S)
Stephen Fitzpatrick and
Cordell Green

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Kestrel Institute
3260 Hillview Avenue
Palo Alto California 94304

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-179

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Robert J. Paragi/IFTB/(315) 330-3547/ Robert.Paragi@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT **(Maximum 200 Words)**
This project addressed the problem of the real-time management of distributed resources. It developed an abstract formulation in terms of distributed constraint optimization and developed a simple, anytime algorithm that was shown to be effective, efficient, robust and scalable to large problems. It extended the formulation and algorithm to the management of distributed sensors in a target detection and tracking application. It also developed high-level models for resource management in time-critical targeting in air campaigns and extended an existing scheduler generator tool to automatically generate executable schedulers that quickly assign aircraft and munitions to prosecute targets of opportunity while minimizing disruption to pre-planned target prosecutions.

14. SUBJECT TERMS
Semi-Automated Program Development, Software Agent Systems, Autonomous Negotiation

15. NUMBER OF PAGES
63

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Table of Contents

List of Illustrations

1. Introduction

This report summarizes Kestrel Institute's achievements in its "e-Merge-ANT" project, part of DARPA's "Autonomous Negotiating Teams" program (ANTs). The over-arching theme of the project was the efficient, autonomous management of distributed resources subject to soft-real-time constraints ("distributed resource management").

In distributed resource management, a set of physical resources – such as sensors or aircraft – are geographically distributed but can communicate, typically using radio. The resources are required to accomplish certain tasks – such as detecting and tracking targets or deploying munitions – and are subject to certain constraints – such as sensing ranges, limited communication, finite speed, and finite energy. In many cases, the resources must collaborate to effectively accomplish their tasks – e.g., several sensors may need to scan a target simultaneously to enable high-quality data fusion, or one aircraft may provide electronic jamming services in support of another.

The problem in distributed resource management is to coordinate the resources to allow them to accomplish their tasks while ensuring that their constraints are respected. In the ANTs program, the problems exhibited strong dynamic aspects: both the tasks to be accomplished and the availability or capability of the resources could vary in real-time. Such dynamics must be accounted for by the resource management mechanism – i.e., it must be adaptive, to continually tune resource behavior to the current circumstances, and robust, to survive the failure of some resources with only proportional degradation in overall performance.

Kestrel's research broadly followed three main lines: (i) distributed constraint optimization; (ii) target detection and tracking in distributed sensor networks; (iii) high-level modeling and automated code generation for time-critical targeting in air campaigns.

For the first two of these, the resource management mechanism itself was distributed (as well as the resources) because the combination of real-time requirements and communication latency for typical application scenarios effectively ruled a centralized mechanism. For time-critical targeting, the resource management mechanism was centralized to allow integration with an existing software architecture (the distributed nature of the resources was manifest as a critical issue to be accounted for in satisfying the real-time constraints).

This last integration requirement notwithstanding, Kestrel's project also focused on characteristics deemed critical by the ANTs program, namely:

- Scalability: the resource management mechanism should be able to scale to very large systems, involving thousands of resources.

- Robustness: localized failures of resources should not cause the failure of the entire system.
- Cost-effectiveness: coordination in a distributed system incurs costs, for computation and communication. These costs may be measured in terms of energy usage (which is important if the resources have limited power supplies) and, for communication, the amount of radio-frequency chatter, that can reveal the location of the resources to adversaries. Consequently, the resource management mechanism should incur low costs where possible.
- Good-enough, soon-enough solutions: resource management problems typically have combinatorial complexity, so that finding optimal solutions may take inordinately long times that are unacceptable given the real-time constraints. On the other hand, heuristic algorithms can often find solutions quickly, but the solutions may be of poor quality. The ANTs program attempted a compromise: the real-time constraints must be observed (the "soon-enough" aspect) but the quality of the solutions, while not optimal, should be better than what might be expected from heuristic algorithms (the "good-enough" aspect).

In the remainder of this introduction, each of the three main lines of research are briefly described. These lines of research are elaborated in detail in later sections.

1.1. Distributed Constraint Optimization

Many aspects of resource management can be expressed in terms of constraint satisfaction or optimization. In constraint satisfaction, each variable in some set is to be assigned a value but the values are subject to constraints that determine the allowable combinations of the values – the objective is to find an assignment that satisfies all of the constraints. Similarly, in constraint optimization, the violation of a constraint incurs a penalty and the objective is to minimize the sum of the penalties.

For example, consider the problem of allowing nodes in a wireless network to use a single radio frequency for communication. To avoid interference, nodes that are within a certain distance should not transmit at the same time. One way to model this problem in terms of constraints is to assign each node in the network a timeslot in a cyclic (periodic) schedule, during which timeslot the node is allowed to transmit. Nodes that are close enough to interfere are connected by a constraint that requires their timeslots to be different. (This is the classic problem of coloring the nodes in a graph so that connected nodes have different colors.)

In *distributed* constraint problems, the assignment of values to variables is to be performed in a distributed manner, by communicating assignment "agents"[1]. This implies an essential locality of responsibility and knowledge in large systems: each agent is responsible for assigning values to some (bounded) subset of the variables; the agent

[1] This term is being used as a convenient name for a component of the distributed assignment mechanism rather for association with the paradigm of "intelligent agents".

knows the values of other variables only by the values being communicated to it and since communication is limited by latency and bandwidth, it is unrealistic to expect any single agent to know the value of every variable as the number of variables grows.

Even though an agent's knowledge and responsibility are essentially local, the effects of an agent's decisions may, in principle, propagate arbitrarily far through the network of constraints; that is, an assignment made to one variable by one agent may require another agent to change its assignment in order to reestablish satisfaction of the constraints affecting its variables; this change in assignment may require a third agent to compensate in turn; and so on. Since communication takes time, it is possible that the overall assignment never stabilizes.

In Kestrel's research, distributed constraint optimization was used as an abstract paradigm for distributed resource management that allowed some essential aspects of practical problems to be isolated from some details that, while important for a particular problem, were probably of narrower interest.

Using this paradigm, Kestrel was able to identify a particularly simple yet effective distributed algorithm based on each agent continually assigning its variables values that minimize the penalties associated with violated constraints, given what the agent knows of the values of other variables, and communicating the assignments to other agents. Kestrel was able to investigate vital aspects of the algorithm's behavior, including convergence, and to develop simple ways to improve its behavior.

1.2. Detection & Tracking of Targets in Distributed Sensor Networks

The ANTs program developed a challenge problem for demonstrating distributed resource management techniques in the context of a distributed sensor network. A set of small radars were used to detect and track moving targets. Each radar was subject to certain constraints and multiple radars were to track each target to improve tracking accuracy. The radars were equipped with radios to allow coordination and data dissemination.

In its work on the challenge problem, Kestrel developed a distributed resource management mechanism based on scan *scheduling*. Each radar continually maintained an estimate of the positions of any nearby targets (based on data it acquired itself and data received from nearby radars). Each radar also continually maintained and communicated to nearby radars a schedule of the scans it planned on taking in the near future. In determining its own schedule, a radar would try to scan nearby targets as well as possible, while taking into account which ones would be scanned by other radars (according to the schedules it had received from the other radars). Thus, the resource management mechanism is a particular instance of the abstract algorithm Kestrel developed in its investigation into distributed constraint optimization.

1.3. High-Level Modeling and Automated Code Generation for Time-Critical Targeting

During the final year of its project, Kestrel investigated a problem proposed by AFRL as part of the TDDE (Time-critical-target Dynamic Decision Enabler) prototype being developed by AFRL and Lockheed Martin Mission Systems. In the context of an air campaign, a time-critical target (TCT) is a surface target that is discovered during execution of a daily battle plan (and was thus not specifically planned for) and that is available for a relatively short duration (maybe 10 to 15 minutes). Such targets often, though not always, have high priority; for example, a typical TCT is a missile launcher that comes out from under cover, is spotted while preparing for a launch and must be destroyed before it goes back under cover.

One of the challenges in engaging TCTs is to identify aircraft that are close enough to the target to be able to reposition to engage it quickly enough, that have appropriate munitions available, have appropriate defensive capabilities (e.g., electronic jamming) and that were not previously assigned a higher-priority target. Given that such aircraft can be identified, their tasking orders should be modified to engage the TCT while minimizing disruption to previously planned actions. For example, if an aircraft is diverted to engage a TCT, it may require air-refueling later to allow it to complete previously assigned missions or to return to base. If a refueling tanker's tasking orders are modified to provide fuel to the diverted aircraft, it may be unable to fulfill previously planned fuel transfers to other aircraft. Thus, a modification to engage a TCT may ripple through existing battle plans.

Kestrel addressed this problem by extending its Planware system for generating schedulers from high-level models of resources and tasks. Planware had previously been developed to support the high-level modeling of batch-oriented transportation scheduling problems. Kestrel extended Planware to support incremental scheduling of target engagements and used Planware to model the TCT problem and generate an appropriate scheduler.

2. Summary of Project Results

Developed notions of real-time, approximate, distributed constraint optimization (DCO).

Showed how DCO applies to distributed resource management.

Developed simple anytime algorithm for DCO.

Demonstrated that the algorithm is scalable, robust and incurs low costs.

Extended DCO notions to distributed resource management in the ANTs challenge problem.

Developed distributed, anytime resource management mechanism for the challenge problem based on an informal notion of expectation maximization.

Developed distributed, multi-target tracker to support resource management in the challenge problem.

Demonstrated effectiveness of the resource management mechanism using challenge problem simulator.

Developed high-level resource and task models for time-critical targeting in USAF air campaigns.

Extended Kestrel's Planware system to automatically generate incremental scheduler for time-critical targeting.

Integrated time-critical-targeting scheduler with components developed by AFRL and Lockheed Martin for broader decision support module for time-critical targeting.

3. Distributed Constraint Optimization

In a constraint optimization problem, each of a set of N variables x_i ($1 \le i \le N$) is to be assigned a value v_i from some domain D_i. The assignment is subject to a set of M constraints c_j ($1 \le j \le M$) each of which assesses a penalty, which may be zero, according to the values assigned to the variables. That is, each constraint is a function from the combined domain of the variables to the non-negative Reals: $c_j \in D \to R^{0+}$ where $D \equiv \prod D_i$. The objective is to find an assignment that minimizes the total penalty $C \equiv \Sigma c_j$. (For satisfaction problems rather than optimization problems, the total penalty is required to be exactly zero.)

In many practical problems, a given constraint will actually depend on only a small number of variables (a constraint *actually* depends on a variable if the value of the constraint can vary as the variable's value varies). Two variables are said to be connected by a constraint if the constraint depends on both of them, and two variables are said to be connected if they are connected by some constraint.

In such a context, it may be useful to view the constraint set as a graph in which each variable is represented as a node and each constraint as a hyper-edge connecting the nodes/variables on which it depends. The standard metrics and classifications from graph theory may fruitfully be carried over. For example, the mean degree of a graph/constraint set is the average number of nodes/variables to which a node/variable is connected, and the density of a graph/constraint set may be defined as the fraction of pairs of nodes/variables that are connected.

For large ANTs problems, it is expected that the constraint set would be sparse (i.e., have low density) because a dense set would implies wide-scale, direct interaction across the entire set of resources, which does not seem realistic as the number of resources becomes large. In fact, it could be argued that a class of constraint problems can be considered scalable only if the mean degree is bounded as the number of variables goes to infinity.

As an example, consider the problem of coloring the nodes in a graph so that each hyper-edge connects only nodes of different colors. If the number of colors is fixed at K then the colors can be represented as integers in Z_K. With X_j representing the variables on which constraint c_j depends, the constraint can be expressed in the form

$$c_j \equiv \Sigma \text{if } n_{jk}>1 \text{ then } n_{jk} \text{ else } 0 \qquad (1 \le k \le K)$$

where n_{jk} is the number of variables that have color k and on which c_j depends; i.e., $n_{jk} \equiv |\{x_i \mid x_i \in X_j \wedge v_i = k\}|$.

It should be noted that the model of random graphs commonly used in graph problems is often inappropriate for constraint problems arising in resource management applications. In random graphs, the probability of an edge connecting two given nodes is independent of any other edges that might connect those nodes to other nodes. However, in resource management problems the probability of a constraint connecting two variables is often higher if there is a third variable to which those two variables are both connected. This correlation arises because interaction between resources is often dependent on their distance in geographical or communication space: for example, two transmitters are more likely to interfere with each other if they are geographically close.

3.1. Distributed Problems

In *distributed* constraint optimization (DCO), the assignment of values to variables is performed by a set of agents that can communicate but only with an inherent latency. In other words, there is an unavoidable delay in propagating information about change. Such problems are natural models for many physical systems in which the distribution of information is inherent, often because the physical devices on which information is stored or generated are physically separated.

In most of Kestrel's research on DCO, it was assumed that there was a one-to-one correlation between agents and variables (i.e., each agent was responsible for exactly one variable), that each agent is aware of the constraints that depend on its variable, and that two agents could directly communicate iff they had variables that were connected. These assumptions do not seem to be significant in principle since, for example it is conceptually straightforward to emulate direct communication using multi-hop communication. In practical terms, though, the extra latency of multi-hop communication may be significant.

3.2. An Anytime, Peer-to-Peer Optimization Algorithm

Kestrel's research in DCO lead to the development of a simple algorithm that is surprisingly effective for many applications. The algorithm is defined below.

1. Initially, each agent assigns its variable some arbitrary value (either chosen at random or computed using some fast heuristic).
2. The agent idles for some random period, during which it may receive information from other agents about the values they have assigned their variables.
3. The agent then assigns its variable a value that minimizes the total penalty assessed by its constraints, with respect to other variables' values known to the agent.
4. The agent then sends the assigned value (if it has changed) to the other agents with which it can communicate.
5. Repeat Steps 2-4 indefinitely.

The main storage and computational costs of the algorithm are incurred in Step 3. Note that efficient methods for finding a value that minimizes the constraint penalty are typically dependent on the types of the values and forms of the constraints. For example, in graph coloring problems, an efficient method is for the agent to maintain a histogram of the use of each color by its neighbors as it receives information from them in Step 2, and to choose any color in Step 3 that the histogram shows is least used.

Nevertheless, in general the costs for a given agent are determined by the degree of the agent's variable rather than by the total size of the constraint set. Thus, if the degree of the constraint is bounded as the number of variables grows to infinity, then the per-variable costs are also bounded; in other words, the algorithm should be scalable to arbitrarily large constraint sets. Similarly, the per-variable communication costs are also bounded.

There are several versions of this algorithm that have nice performance under particular circumstances. For example, in loosely constrained systems – i.e., one in which there are many solutions for a given constraint set – a "conservative" version – in which an agent changes its variable's value only if the current value gives rise to a non-zero penalty – can perform better than the standard version shown here. Other variants can be designed for synchronous systems (in which the agents operate in well-defined rounds).

3.3. Experimental Assessment

This algorithm has been experimentally assessed using standard problems (such as graph coloring and leader election) and random problems, using sparse constraint sets and dense sets, and using loose and tight constraint sets. The following general characteristics were observed.

- Anytime improvement: the quality of the initial solution is determined by whatever method is used to generate the solution – random or heuristic. As the algorithm iterates, the quality of the solution initially quickly improves, and then asymptotically converges. Often, the asymptote is close to the theoretical best or the best found using a centralized algorithm. (Note, the quality of the solution is assessed globally using all of the variables' values.)
- Scalability: in the preceding section, it was noted that the algorithm's per-variable *costs* are bounded as the number of variables goes to infinity. Experiments also show that the solution *quality* is maintained.
- Adaptivity: by varying the node and constraint set at random while the algorithm is running, it was shown that the algorithm quickly adapts to change, even sudden, large-scale change.
- Robustness: by corrupting and dropping messages, it was shown that the algorithm is robust against communication error.

Experiments also showed that the algorithm could be pushed into a state called "thrashing", in which a large fraction of the agents are continually changing their variables' values with no overall improvement in solution quality. In extreme cases, the solution quality was lower than what would be expected from random assignment of values to the variables.

Thrashing is caused when the idle period (Step 2 of the algorithm) is too low compared with the communication latency: if agents change values quicker than the changes can be communicated, then the information on which agents based their optimization decisions (Step 3 of the algorithm) is always out of date, so the agents make poor decisions.

Fortunately, and somewhat surprisingly, it was found that moderate values for the idle period worked well across a wide range of constraint sets, regardless of how tightly constrained or how dense.

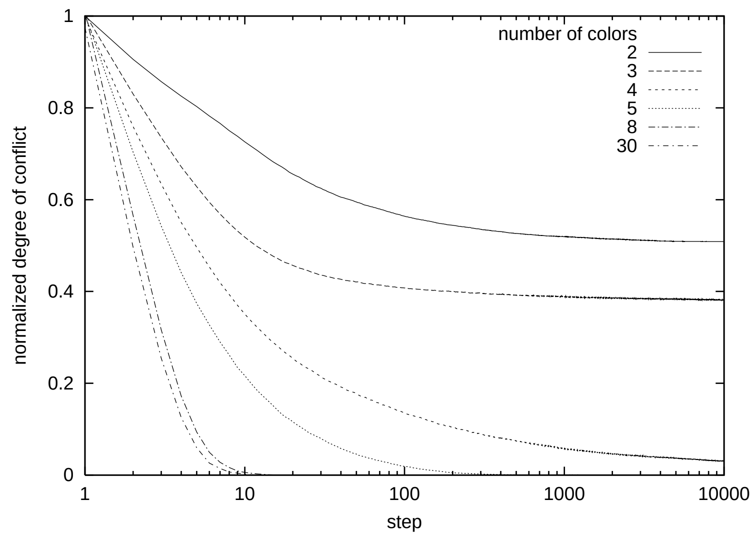

Figure 1: Typical performance of the algorithm for coloring a graph with various numbers of colors (2 and 3 colors represent a tightly constrained system, while 8 or more colors represent a loosely constrained system). The vertical axis measures the penalty incurred for constraint violations while the horizontal axis measured the number of iterations each node makes through the idle-optimize-communicate process.

4. Detection & Tracking in Distributed Sensor Networks

The ANTs challenge problem was designed as a room-sized demonstration platform for distributed resource management that incorporated real-world concerns such as sensor noise and communication limitations. The objective was to detect and track moving targets using small, radio-equipped Doppler radars, each of which could scan a single target, at any given time, within a limited distance. The resource management aspect of the challenge problem was to coordinate the radars to ensure that targets were quickly detected and subsequently scanned continuously by several radars (to achieve high-quality tracking) as they moved into and out of range of individual radars.

Each radar sensor consisted of three emitter/detector pairs, oriented at 120 degree intervals, only one of which can be scanned at any given time, with each scan taking up to 0.6 seconds for an amplitude-only measurement or 1.8 seconds for an amplitude-and-frequency measurement. Using standardized targets, amplitude readings gave estimates of distance while frequency readings gave estimates of radial speed (towards or away from the radar).

Figure 2:BAE radar showing three emitter/detector pairs

Each emitter could be independently activated or deactivated. While an emitter was active, it consumed power, regardless of whether or not its detector was being sampled. If a deactivated emitter was activated, the emitted beam was unstable and did not give reliable measurements for approximately 2 seconds.

The strength of the signal reflected off a target was determined by the following equation:

$$m(R,\theta) = \exp(-(\theta/A)^2) \,.\, K/R^2$$

where R was the sensor-target distance, θ was the angle between the emitter's mid-beam and the target (0 degrees $\le \theta <$ 180 degrees), and K and A were constants. The constant A was about 40 degrees so that, for example, a target that was at mid-beam would have produced a signal that was $e^{9/4} \approx 9.5$ times larger than a target that was at the same distance from the sensor but 60 degrees off mid-beam. The emitters, detectors and sampler were subject to random noise (both internal and environmental) that limited the effective range of the sensors to about 7 meters.

If a radar beam reflected off several targets simultaneously, the reading produced by the detector were essentially a random combination of what the individual targets would produce separately, and was effectively useless. Each radar sensor had an omni-directional radio transmitter and receiver that could be used to communicate measurements and arbitrary data in short bursts (message lengths preferably below 100 bytes). The communication range was limited by the effective physical transmission range, so its "broadcast" mode is really a local, multicast mode.

A single radar amplitude measurement corresponded to a contour in (R,θ). To accurately locate a target, more-or-less simultaneous measurements from several, nearby sensors had to be combined using trilateralization. Of course, readings from sensors are subject to noise so new measurements for a target were combined with the target's track to achieve the best compromise between where the target was expected to be (based on its history) and where the sensors reported it to be.

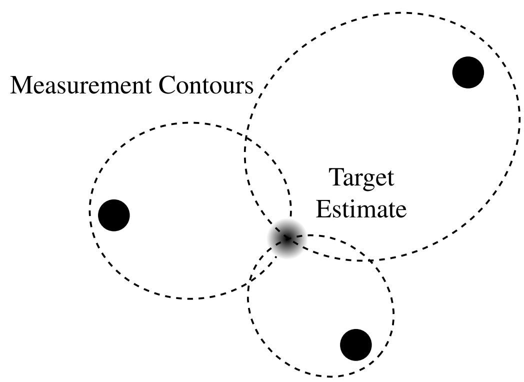

Figure 3:Estimating a target's position from the measurement contours of three radars

4.1. Solution Design

Kestrel's solution for the challenge problem had two main components:

1. A distributed, multi-target tracker that ran on each radar to maintain "world estimates" representing estimates of the positions and velocities of nearby targets.
2. A distributed scan scheduler that ran on each radar to coordinate the radar's scans with those of nearby radars, with the objective of producing high quality scans of known nearby targets.

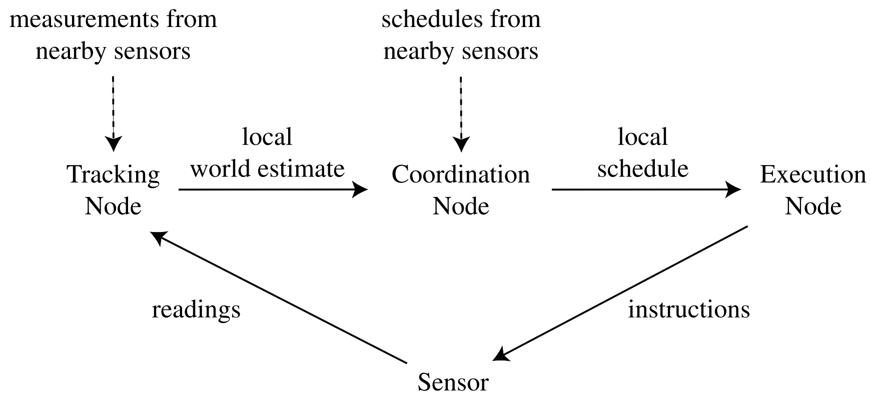

Figure 4:Architecture of the component running on a single radar

The distributed, multi-target tracker was built using the centralized, single-target tracker modules provided by BAE and the University of South Carolina – it was not a focus of Kestrel's work. The distributed scheduler is described in the following sections.

4.2. Distributed Scheduler

The distributed scheduler operated using the same principles as described for distributed constraint optimization: each radar had a scheduling agent that was solely responsible for producing that radar's scan schedule. Each agent continually executed to continually adapt the scan schedules to changing circumstances, with randomized idle periods to ensure coherence. When updating a radar's scan schedule, the radar's scheduling agent took into account the scan schedules of other, nearby radars (as communicated to it) and the estimates of nearby targets (derived from the distributed tracker).

The interesting aspect of this particular application of the distributed constraint optimization algorithm was how to optimize a radar's scan schedule. A scan schedule was represented as a finite map from contiguous time slots to sector numbers, where a sector number identified one of the three emitter/detector pairs. Given a world estimate and existing scan schedules for its radar and nearby radars, a scheduling agent could predict where targets were likely to be during each time slot and compute a quality metric that represented how well those locations would be scanned by its own radar and nearby radars (the details of this quality metric are presented below).

This quality metric was the basis for a hill-climbing algorithm in which the scheduling agent transformed individual time slot-sector assignments for its own radar to try to improve the overall scanning quality.

4.2.1. Scanning Quality Metric

Given a world estimate and a set of radar scan schedules, there were two main terms in computing an overall scan quality metric:

1. The "target density field" T. In principle, this was a function from space-time coordinates to the Reals that represented how many targets were expected to be found

at that point (in space and time). In practice, T was computed over a discrete grid of (x,y,t) points representing 2-dimensional space and time.

2. The "scan quality field" S. This represented how well each point (in space and time) was being scanned, taking into account the cumulative effect of the radars. As above, in principle S was considered to be a function on continuous space-time coordinates; in practice, it was a map over the same discrete grid in (x,y,t).

The overall metric was the inner product of these two fields (i.e., they were multiplied point-wise and them summed); consequently, the metric awards high scores to scan schedules that result in good scanning of space-time points where targets are likely to be.

4.2.1.1. Target Density Field

Ideally, the target density field would be derived from probabilistic estimates over target trajectories derived from the tracker. However, the basic tracking modules did not provide probabilistic estimates so instead an estimate was made as to the relative importance of each point in space-time, based on two main terms:

1. Target estimates. It was assumed that the number of targets remains fixed during one hill-climbing step. A discrete estimate was available for each target. Each target estimate was comprised of a time stamp, a single position estimate and a single velocity estimate. The trajectory of a given target was predicted by linearly extrapolating from its estimated position using its estimated velocity. Such a trajectory would, of course, be a zero-width line; a nominal fuzziness representing uncertainty was imposed on the trajectory (with the amount of fuzziness increasing the further out in time being predicted).

2. Raw measurements. Raw radar measurements could provide reasonably certain information about limited regions of space over limited time periods: if a strong radar measurement was obtained, it was reasonably certain that a target was present in the region of space corresponding to the measurement's sector when the measurement was taken; otherwise, if a measurement was essentially background noise, then it was reasonably certain that no target was present. Thus, for a given measurement, a positive or negative value could be added to the overall target density field for each space coordinate in the corresponding sector, at the time of the measurement. For other times, the value added could be reduced in magnitude to account for how our knowledge degrades (since a target may not stay in given sector).

The rationale for using these two factors was that raw measurements provided information that was reasonably reliable but limited in scope (in space and time) whereas target estimates provided information that was easy to extrapolate but that had proven to be somewhat unreliable in previous experiments.

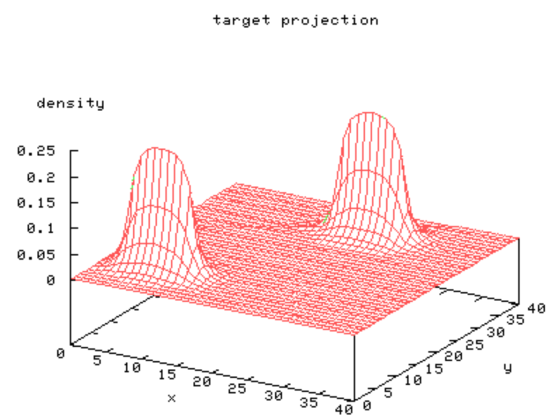

Figure 5:Predictions of the likely locations of two targets averaged over time. One target starts at the bottom left (near the origin) and moves to the right; the other starts at the top right and moves to the left.

4.2.1.2. Scan Quality Field

The scan quality field represents how well each point (in space and time) is cumulatively scanned, under the following informal rules:

- a scan by a given head on a given radar unit will produce good data for a target that is close to the unit and near the head's mid-beam;
- fewer than two different radars scanning the same point simultaneously produces low scan quality;
- scanning by two or three different radars produces good scan quality;
- scanning by more than three produces scan quality that is higher, but only marginally so.

The overall scan quality field was computed by first computing how well each point (in space and time) was scanned by each individual radar, and then computing the cumulative effect.

4.2.1.3. Single-Radar Scan Quality Field

How well a given radar would scan each point in space and time was determined by the measurements that it was to take. Ignoring initially the temporal aspects, the quality of scanning at some point in space was taken to be a reasonable function of the strength of the radar signal that would be detected if a target were at that point:

$$\text{quality}(R,\phi) = \log(\text{signal}(R,\phi))/\log(\text{max\_signal})$$

where R was the distance to the point, ϕ was the angle between the point and the mid-beam of the radar head taking the measurement, signal(R,ϕ) was the standard radar model of the signal strength and max_signal was the maximum signal that could be observed (approximately 4000).

Now consider the temporal aspects. A single radar scan would measure a given point in space over some non-zero time period – information would be acquired about any target that was present at that point in space at any time during the measurement period. However, it seems reasonable that the quality of information obtained would be higher the longer the target was present. It thus seems reasonable to associate a high quality with the mid-point of the measurement period and somewhat lower quality with the start and end points.

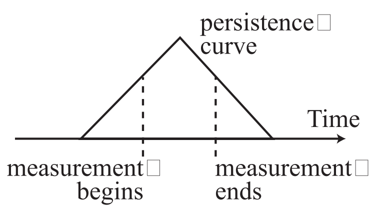

Figure 6:Quality of information from a measurement as a function of time

Moreover, since targets would tend to have non-zero spatial extent and to move at finite speeds, a non-zero quality was associated with time points outside the actual duration of the measurement. (For example, if a measurement lasting two seconds is taken every three seconds then some knowledge about the time in between measurements is expected to be acquired.)

These factors were modeled by awarding a high quality of scanning to a measurement's mid-point and linearly decaying the quality on either side. The overall quality field arising from multiple measurements (by the same radar) was computed by summing the quality fields arising from the individual measurements.

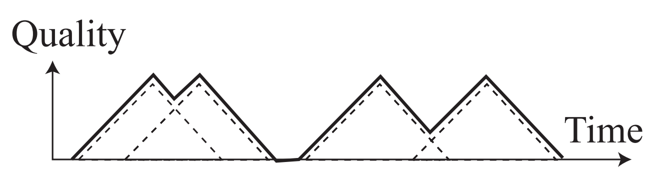

Figure 7:Quality of information from a sequence of measurements from a single radar, as a function of time

4.2.1.4. Multiple-Radar Scan Quality Field

To compute the cumulative effect of scans by multiple radars, a non-linear "amplification" function was used that awarded: low scores when its input was equivalent to about one strong radar measurement; moderate scores for input equivalent to two strong measurements; high scores for three strong measurements; and only slightly higher scores for four or more strong measurements.

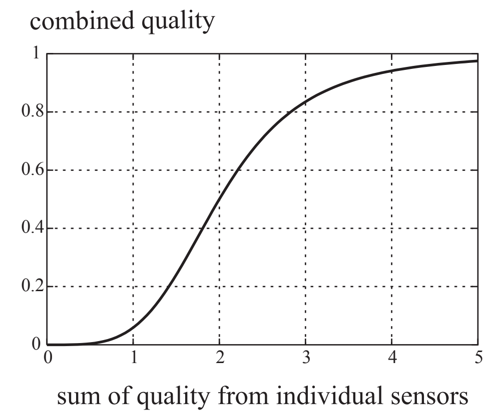

Figure 8:Non-linear amplification function representing the cumulative quality of scans from multiple radars

When combined with the cost of taking measurements, such a function would encourage collaboration on scanning targets but discouraged swamping of targets with excessive sensor energy, since the differential in the quality-cost tradeoff would become negative at high sensor energies.

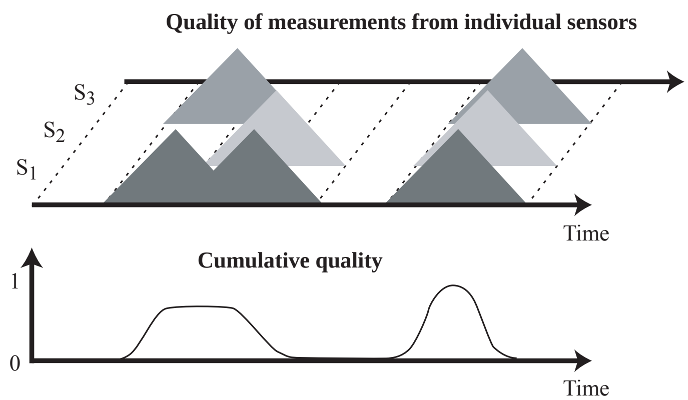

Figure 9:Overall quality of scanning, as a function of time, from sequences of measurements from multiple radars

4.3. Experimental Assessment

Kestrel's distributed coordination mechanism was evaluated using the Radsim simulator for the challenge problem developed by AFRL. Typical results are shown below. There were 8 sensors, indicated by the groups of triangles (indicating the 3 emitter-detector pairs per sensor). There were two targets, moving in oval tracks, indicated by the unbroken lines, around a room of dimensions 40 units by 40 units.

The black dots represent typical output from a tracking node: each dot is an estimate of a target's position. The error in a position estimate p at time t can be computed as the distance between p and the true target position at time t. The track quality can be measured as the root-mean-square of the single-estimate errors.

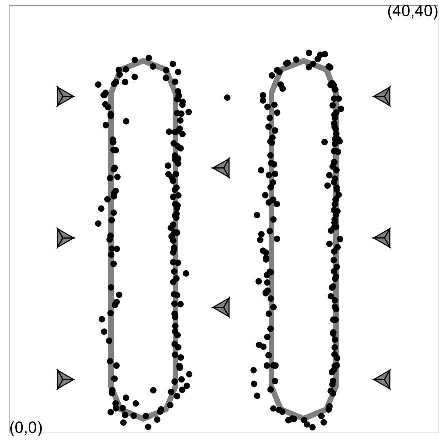

Figure 10:Tracking estimates for two targets

For a single target (not illustrated), r.m.s. errors of about 0.6 units were obtained; for two targets, about 0.9 units. These values were close to the limit (about 0.5 units) of what the tracker was expected to be able to achieve, given its approximation techniques, its model of the targets, and (simulated) sensor noise. They also compare well with errors obtained using purely random choice of sectors to be measured, namely about 2.2 units for a single target.

The mean power usage was about 50% (meaning that, on average, half the emitters were on at any given time). This is slightly higher than was hoped for, especially for a single target, for which it should have been possible to reduce the power usage to around 33% (meaning that only one emitter would be active on each sensor, on average).

Evaluation was also carried out using hardware. These results were much less satisfactory. Several problems were identified:

- The sensor signal model differed significantly from the theoretical model, so processing the data in the tracker was problematic. The final challenge problem demonstration attempted to ameliorate this problem by combining, in a controllable fashion, real signals from the hardware with artificial signals generated from the theoretical models (given the target's true position). This indubitably would have improved the results, had not further problems intervened.
- Towards the end of the final demonstration, Kestrel determined that its multi-target tracker was sending too many message, causing many of them to be lost due to interference. Using a lossless communication emulator seemed to result in better tracks.

5. High-Level Modeling and Automated Code Generation for Time-Critical Targets

This section presents Kestrel's overall approach to time-critical targeting and its high-level models of resources and targets. The following section presents the Planware tool that is used to formalize these models and to automatically generate scheduling code from the formalization.

The time-critical-targeting Dynamic Decision Enabler (TDDE) is a prototype component being developed by AFRL and Lockheed Martin as a candidate for insertion into the Theater Battle Management Core Systems (TBMCS). TBMCS is a key command and control system supporting USAF air campaigns. TDDE is designed to augment TBMCS by supporting rapid decision making regarding the prosecution of surface targets of opportunity – that is, targets that are identified during pre-planned operations and that are prosecutable for relatively short periods (tens of minutes, typically).

Three significant aspects in the TDDE decision process are:

- Weapon-target pairing: selection of appropriate resources to prosecute the TCT. These resources are typically aircraft or missiles.
- Threat assessment: identification of threats that may be encountered by selected aircraft while prosecuting the TCT (including during ingress and egress).
- Support package creation: selection of appropriate resources to support the primary resources in prosecuting the TCT. These resources are typically electronic warfare aircraft, fighter escorts and refueling tankers.

In general, these aspects should be addressed as a joint scheduling problem, since they are interdependent. For example, threat assessment cannot be carried out until weapon-target pairing has been performed, since the threats expected to be encountered depend on the aircrafts' flight paths. And support package creation cannot be carried out until threat assessment has been performed since the type of support required depends on the types of threat expected. But weapon-target pairing cannot be completed until support package creation has been performed since a given choice of aircraft cannot be confirmed until appropriate support has been selected.

Kestrel addressed the weapon-target pairing and support package creation aspects as a joint scheduling problem, utilizing a threat assessment component developed by AFRL. This represented a significant advance over other weapon-target pairing packages which attempted to linearize the joint problem into three successive, one-off decisions, which frequently resulted in no feasible solution being found.

Kestrel's scheduler has been delivered to Lockheed Martin and AFRL and should be included in a demonstration in the latter half of 2004.

5.1. Overview of Kestrel's Approach to TCT Prosecution

Kestrel's Planware application is a tool for modeling scheduling/planning problems at a high level and automatically generating executable scheduling/planning code for specific problems. For example, to model TCT prosecution, significant resources such as aircraft, munitions and missiles are described in terms of what actions they can perform and what constraints they must obey (e.g., aircraft can reposition and deploy munitions; they are subject to maximum flying speeds and munitions capacities; munitions have characteristics such as how far from a target they can be released and how big an explosion they cause). Likewise, tasks that are to be carried out are described; for example, a target description might include the target's position, what type of armor it has and how much collateral damage is allowed.

Planware combines such descriptions with generic scheduling/planning algorithms to create highly tailored scheduling/planning code for the specific problem described. This code can be compiled to produce a stand-alone executable scheduler/planner.

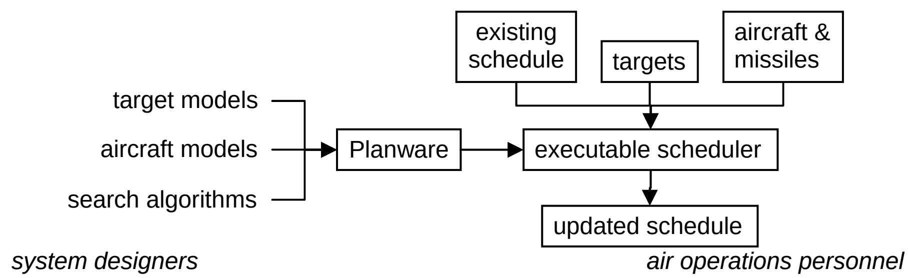

Figure 11:Planware generates executable schedulers from high-level models

It is the scheduler/planner that would be an active run-time component of TDDE & TBMCS (Planware being a design-time component). Due to legal and security restrictions, integration with the TBMCS system was accomplished through XML-based exchange of data files rather than through direct access to the TBMCS databases. Consequently, the scheduler takes as input an existing schedule (as well as target and resource information) and returns a modified schedule which will include the prosecution of any TCTs if appropriate resources could be found.

5.2. General Problem Statement

The general problem to be addressed may be formulated as follows. The *Red Force* has under its command various ground vehicles and installations, which are potential targets for the Blue Force. Ground vehicles include tanks, howitzers and mobile missile launchers; installations include buildings, camps, munitions dumps, radars, defensive installations (e.g., anti-aircraft guns and surface-to-air missiles), roads and bridges.

The *Blue Force* has under its command various aircraft. *Strike aircraft* are capable of destroying Red targets, when equipped with appropriate munitions; *support aircraft* include some capable of destroying some classes of Red targets (e.g., radar and defensive installations), electronic warfare aircraft and refueling tankers.

Blue aircraft have been assigned to engage Red targets over periods of one to three days. Any given aircraft is capable of engaging only a limited number of targets because of various *constraints*, which include finite munitions capacity, finite fuel capacity and finite speed.

During engagement, new Red targets are identified as *Time-Critical Targets* (TCTs) which can be engaged for only a limited time. The objective is to adjust the assignment of Blue aircraft to optimally engage the Red targets, including the newly-identified targets. The quality of an assignment may be gauged in terms of how many TCTs are engaged and how many of the originally-engaged targets are still engaged, taking into account any disruption caused by engaging the TCTs. Disruption occurs when engaging a TCT requires an aircraft to forego engaging some other target because of the aircraft's constraints.

Since time is of the essence for TCTs, the reassignment must be computed quickly, say within one minute – strict optimality may be sacrificed to achieve quick computation.

This problem is formulated more precisely in the following sections.

5.3. Representing Red Target Information: Target Tracks

Intelligently assigning Blue aircraft to Red targets requires some knowledge about where the Red targets are and where they are likely to go. In this problem formulation, it is assumed that each target is expected to follow a piece-wise linear trajectory: that is, at times T_0, T_1, …, T_n, the target is expected to be at positions p_0, p_1, …, p_n (for some n) and the target is expected to move with a constant velocity between each position. (Note that a stationary target, such as a building, can be represented with a degenerate trajectory in which all positions are the same.)

Additional information may be associated with each target:

- Class: a designation of the type of target (e.g., Tank). This information can be used, for example, to determine the effectiveness of particular classes of munitions.
- Value: the importance of this target relative to other targets. In this formulation, a target's value is represented as a real number having absolute significance: thus, for example, 1 target of value 11 is considered more important that 2 targets each of value 5.
- Collateral damage radius (optional): the maximum distance from the target at which collateral damage from munitions is permitted. This radius may be used to determine if a particular class of munitions is permissible.
- Approach angle (optional): the direction from which a Blue aircraft must attack the target.

Some of this information may be specified for individual segments of a target's trajectory; for example, the collateral damage radius may be reduced while the target is passing civilian buildings.

5.4. Representing Blue Aircraft Information: Aircraft Schedules

Blue aircraft are to be assigned to directly engage Red targets and to support such engagements. In this problem formulation, assignments are represented on a per-aircraft basis.

An aircraft's assignments are represented as a *schedule*, in which particular *actions* are to be performed over specified *time periods*. Typical actions include:

- flying to specified coordinates;
- engaging a specified target (which may require fixing the target and deploying munitions);
- rendezvousing with a specified Blue aircraft at specified coordinates (e.g., a strike aircraft may rendezvous with a refueling tanker or a support aircraft may rendezvous with a strike aircraft);
- refueling;
- jamming Red sensors;
- destroying Red radar or defensive installations to support a strike mission.

Note that some actions may occur simultaneously (e.g., flying and jamming) and some actions require additional information to be specified (e.g., with what type of munitions a target is to be struck).

Given an aircraft's schedule (and some information about the initial state of the aircraft), other useful information can be computed; for example, the aircraft's trajectory or the dynamic state of consumable resources (e.g., fuel and munitions) at any given time. Some actions may be indeterminate in outcome or in resources required; e.g., the time required to fix a target may not be known in advance, or the number of Red installations destroyed by support aircraft – and thus the number of missiles deployed – may not be known in advance. In this problem formulation, appropriate upper- or lower-bounds will be used for indeterminate quantities.

5.5. High-Level Constraints on Schedules

The actions of an aircraft are constrained due, for example, to such physical limitations as fuel and munitions capacities and maximum speed. These constraints carry over to aircraft schedules: for example, the time allowed in a schedule for an aircraft to fly between two positions must not be so short as to require the aircraft to exceed its maximum speed.

Other constraints arise from the interactions between multiple aircraft or between a Blue aircraft and a Red target. For example: if an aircraft and a refueling tanker are scheduled to rendezvous, they must be scheduled to be at the same coordinates at the same time; if an aircraft is scheduled to engage a target at some time, the scheduled engagement position must be within striking range of the target's (expected) position at that time.

Because the Blue aircrafts' actions are explicitly scheduled and the trajectories of known Red targets are (to some degree) predictable, constraints such as the above can be expressed as predicates over schedules and target tracks. For example, the maximum speed constraint can be expressed as follows:

$$\forall S{:}\mathrm{Schedule},\ f{:}\mathrm{Action} \in \mathrm{flights}(S) \cdot$$
$$\mathrm{duration}(f) \times \mathrm{maximumSpeed}(\mathrm{aircraft}(f))$$
$$\geq \mathrm{distance}(\mathrm{origin}(f),\ \mathrm{destination}(f))$$

This can be read as follows:

- For all schedules, denoted by *S*,
- and for all actions, denoted by *f*, contained in *S* that correspond to flights (rather than, say, engagements),
- the maximum distance that the aircraft could fly in the time allowed must be greater than the distance of the flight.

When an aircraft is scheduled to engage a newly-identified TCT, the parameters of already-scheduled actions may need to be adjusted to accommodate actions introduced for the new TCT. For example, the duration of a scheduled flight may need to be reduced to allow time for the aircraft to intercept and engage the TCT. Such a reduction may cause the maximum speed constraint to be violated, indicating that a choice may need to be made between engaging the TCT with this aircraft or dropping the scheduled flight and any other actions that depend on it (such as engaging other targets at the destination). Such a choice would be informed by the *objective function* which would take into account the importance of the TCT versus the total importance of scheduled targets that would be dropped.

In general, the constraints on the aircrafts' schedules introduce dependencies between scheduled actions, and when new actions are added or scheduled actions are modified or dropped, the dependencies propagate ripple effects through the schedules. Some of the ripple effects may be suppressed by utilizing slack in the schedules (e.g., an aircraft may be able to increase its speed in earlier flights to allow a new flight to be introduced) but some may require disrupting schedules (e.g., dropping scheduled targets).

5.6. Rôle of Support Aircraft

Support aircraft fulfill rôles that are necessary or helpful to the primary engagements scheduled for strike aircraft. Support aircraft are explicitly scheduled and have their own particular constraints. Some of these constraints relate to the physical limitations of the aircraft (e.g., maximum speed and maximum fuel capacity) and some relate to their interaction with strike aircraft (e.g., EW aircraft must be at approximately the same location at the same time as the strike aircraft they are supporting). Such constraints can be expressed in a similar manner as those for the strike aircraft.

Many of the actions performed by support aircraft are intended to reduce the threat to strike aircraft posed by Red defensive installations by, for example, jamming or destroying Red radars and surface-to-air weapons. The locations and classes of known

Red installations can be used to annotate segments of the strike aircrafts' trajectories with information about the threats posed by the Red installations. Such *threat assessments* may indicate the likelihood that a Red installation may prevent the successful accomplishment of Blue engagements or may endanger Blue strike aircraft. Support aircraft can be scheduled to precede or shadow the strike aircraft through those segments to suppress the threat; for example, offensive, support aircraft may be scheduled to destroy Red SAM sites before strike aircraft fly near their locations or EW aircraft may be scheduled to accompany strike aircraft while they fly near Red SAM locations.

5.7. High-Level Objective

It is expected that the number of Red targets may exceed the number that could possibly be engaged by the Blue forces in the time under consideration. Consequently, decisions may need to be made as to which targets should be engaged. Engagement decisions are informed by such factors as the value of the targets and the effectiveness and permissibility of various types of munitions against a variety of targets. (For example, by loading one type of munitions, an aircraft may be able to engage a small number of high value targets, whereas by loading another type of munitions, the aircraft may be able to engage a high number of low value targets.)

Given an aircraft's schedule, the *target engagement score* may be computed by considering each scheduled target engagement and computing an effective score for that target, and summing over all engagements. The effective score for a single engagement would reflect the value of the target, the effectiveness of the munitions the aircraft is scheduled to deploy against the target, and the likelihood of successful engagement (which would be affected by such factors as suppression of Red defenses by support aircraft). For multiple aircraft, the target engagement scores may be summed to compute a total score for all of their schedules.

To a first approximation, **maximizing the total target engagement score may be taken as the objective in scheduling the Blue aircraft**.

Of course, scheduling must ensure that basic constraints such as the maximum speed constraint are observed. In addition, other terms may be taken into consideration:

- Minimizing the disruption to pre-scheduled engagements may be given explicit emphasis by using different weights for non-TCT targets whose engagements are dropped and for TCTs that are added to schedules.
- The *cost* of the scheduled actions should be minimized, where the cost may be measured, say, in terms of the amount of fuel and munitions scheduled to be consumed.
- The operational risk of the actions should be minimized, where the risk may be measured, say, in terms of damage Red forces might inflict on Blue aircraft and likelihood of successful engagement of targets.

It may be appropriate to take as the overall objective function a linear weighting of the total target engagement score and these terms.

5.8. Red Target Models

All Red forces – such as vehicles, installations and buildings – that are of interest are modeled as *targets*. In any given scenario, a specific target may or may not be designated to be attacked; even if a target is not to be attacked, it may still be relevant for, say, threat assessment.

The track database in the air theater management system is expected to provide the following information for each target:

Target::

id:	TargetDesignator
class:	TargetClass
value:	Real
minKillProbability:	Real
explosives:	ExplosiveYield
airThreatRange:	Distance
trajectory:	Trajectory

The *target designator* is an uninterpreted symbol that uniquely identifies a target. The *class* indicates the type of target (tank, SAM, etc). The *value* indicates the target's importance.

The *minimum kill probability* determines the classes and quantities of munitions that should be deployed against the target – their combined probability of destroying the target must exceed the minimum kill probability. However, due to a lack of publicly available data on munitions effects, a simplified formulation is used here: the *explosives* property specifies a total weight of explosives that must be deployed against the target (*ExplosiveYield* is a weight in kilograms).

The *air threat range* indicates the distance over which a target may be a threat to aircraft.

The *route* indicates where the target is expected to go, in terms of a sequence of way-points between which the target moves with constant velocity:

Trajectory:: sequence of WayPoint
WayPoint::

position:	Coordinates	(3-dimensional)
time:	Time	(absolute date and time)
angleOfAttack:	PlanarAngle	(degrees, optional)
damageRadius:	Distance	(meters, optional)

Each way-point indicates where the target is expected to be at the indicated time (note that 3-dimensional coordinates are used, even though the target is a ground target, in case altitude is a consideration). It may also indicate a direction from which the target must be attacked and a maximum collateral damage radius: these two properties apply to the target until the next way-point.

It is assumed that there is no value in engaging a target before the time of the first way-point or after the time of the last way-point. A stationary target may be specified using two way-points that have the same position: the two specified times would determine the period during which the target should be attacked.

5.8.1. Example Target Classes

The following classes of target are used in this problem formulation:

- Surface-to-Surface Missile launcher (SSM):
  light explosives required; value 10; range 120 km.
- Surface-to-Air Missile launcher (SAM):
  light explosives required; value 15; air threat range 60 km.
- Anti-Aircraft Artillery guns (AAA):
  light explosives required; value 5; air threat range 4 km.
- Early Warning Radar (EWR):
  medium explosives required; value 14; air threat range 100 km.
- Target Acquisition Radar (TAR):
  medium explosives required; value 15; air threat range 10 km.
- Armored Personnel Carrier (APC):
  medium explosives required; value 0.1; travel range 500 km, speed 65 km/h.
- Self-propelled Howitzer (HOW):
  light explosives required; value 0.2; munitions range 25 km; travel range 250 km; speed 55 km/h.
- Tank (TNK):
  medium explosives required; value 0.4; munitions range 30 km; travel range 450 km; speed 60 km/h.
- Depot (DP):
  value 0.1.
- Building (HQ):
  value 0.3.
- Bunker (BNK):
  value 0.2.
- Bridge (BRG).

Target values may change depending on circumstances; for example, weapon stores may be more valuable during the sustained phase of a campaign than during the initial phase.

For the purposes of examples, “light” explosives may be taken to be 50 kg, “medium” 100 kg, and “heavy” 200 kg. For the last four target classes, explosive requirements are determined on a per-target basis.

Only SAMs and AAAs are capable of damaging aircraft, although EWRs and TARs are also considered threats to aircraft. The speed and travel range characteristics are not currently used in formulating the problem but may be useful for constructing demonstration scenarios.

5.9. Blue Aircraft Models

In this problem formulation, all Blue forces are aircraft – ground resources such as landing strips are not considered. For each aircraft of interest, the following information is assumed to be available:

Aircraft::

id:	AircraftDesignator
class:	AircraftClass
schedule:	AircraftSchedule

The *aircraft designator* is an uninterpreted symbol that uniquely identifies an aircraft. The *class* indicates the type of the aircraft (fighter, bomber, EW, etc). The *schedule* indicates what actions the aircraft will be performing – see below.

In this problem formulation, the following classes of aircraft are considered: Strike-Support Fighter (SSF), Ground Attack (GA), Refueling Tanker (RT) and Electronic Warfare (EW). Each class of aircraft has certain properties, some of which may be shared with other classes. For simplicity, in this formulation, all aircraft are modeled using the following, common properties, some of which are optional:

maximumSpeed:	Speed	(meters/second)
fuelCapacity:	FuelQuantity	(kilograms)
refuelingCapacity:	FuelQuantity	(kilograms, optional)
refuelingDuration:	Duration	(seconds)
refuelingProximity:	Distance	(meters)
burnModel:	FuelBurnModel	
standoffEWRange:	Distance	(meters, optional)
escortEWRange:	Distance	(meters, optional)
EWCapacity:	Natural	
munitionsCapacity:	MunitionsCapacity	(optional)
rearmingDuration:	Duration	(seconds, optional)
engagementDuration:	Duration	(seconds, optional)
defenseRange:	Distance	(meters, optional)
defenseCapacity:	Natural	
minOnGroundDuration:	Duration	

The *maximum speed* is taken to be a constant: in subsequent formulations, it may be modeled as being dependent on the aircraft's loaded weight or external conditions (e.g., wind speed).

The *fuel capacity* is the maximum amount of fuel the aircraft can carry for its own use; in contrast, the *refueling capacity* refers to the amount of fuel the aircraft can carry for refueling *other* aircraft. The default value of the refueling capacity if none is explicitly provided is zero.

The *refueling duration* is the maximum time required for the aircraft to refuel. The *refueling proximity* is the maximum distance a refueling tanker can be from a client aircraft when *beginning* refueling.

The *fuel burn model* has two components that predict (i) the amount of fuel consumed in covering a specified distance in a specified time and (ii) the amount of fuel consumed by an airborne aircraft holding its position for a specified period (i.e., circling to maintain an approximately constant position):

FuelBurnModel:: Distance × Duration → FuelQuantity	(meters × seconds → kilograms)
FuelBurnModel:: Duration → FuelQuantity	(seconds → kilograms)

The *standoff EW range* indicates over what distances an EW aircraft can effectively cloak client aircraft against enemy threats from a standoff position. The *escort EW range* indicates how close an EW aircraft must remain to a client aircraft to effectively cloak it against enemy threats during escort. The *EW capacity* indicates the maximum number of simultaneous threats against which an EW aircraft can provide effective protection. The default value if none is explicitly provided is zero, indicating that the aircraft cannot provide EW support.

The *munitions capacity* indicates how many of various types of munitions the aircraft can carry. In this problem formulation, the munitions capacity is abstractly modeled as a predicate that specifies allowable *munitions loads*, where a munitions load specifies a quantity (possibly zero) of each class of munitions:

MunitionsCapacity:: MunitionsLoad → Boolean
MunitionsLoad:: MunitionsClass → Natural

The default value if none is explicitly provided is a munitions capacity that allows no munitions to be loaded (i.e., the aircraft cannot carry munitions).

The *rearming duration* is the maximum time required to rearm this class of aircraft. The default value of zero applies when the aircraft has no offensive capabilities.

The *engagement duration* is the maximum time allowed for the aircraft to fix and attack targets. The default value of zero applies when the aircraft has no offensive capabilities.

The *defense range* is the maximum distance the aircraft can be from client aircraft during escort. The default value of zero applies when the aircraft has no defensive capabilities.

The *defense capacity* is the maximum number of *simultaneous* threats against which a support aircraft can feasibly protect client aircraft.

The *minimum on ground duration* is the shortest time over which the aircraft can be scheduled to be on the ground (taking into consideration how long it requires to land and take-off).

5.9.1. Munitions Models

In this problem formulation, there are four main considerations regarding munitions: maximum distance at which the munitions can be deployed against a target, how a particular class of munitions affects a particular class of target, the likelihood of hitting a target, and the range over which collateral damage may occur. Also of interest are the explosive yield (it is used as a substitute for the effects model) and the per-unit cost.

MunitionsClass::

standoffRange:	Distance	(meters)
effectModel:	TargetClass → KillProbability	
hitProbability:	Real	(probability)
yield:	ExplosiveYield	
damageRadius:	Distance	(meters)
unitCost:	Real	(U.S. dollars)

The *standoff range* is taken to be a constant in this problem formulation; it may be necessary to include dependencies on the class of aircraft deploying the munitions and the class of target.

The *effect model* indicates the effectiveness against particular classes of target and is modeled as a probability of destroying such targets. However, in this problem formulation, a simpler characterization of munitions effectiveness is used: each class has an associated *explosive yield* (weight of explosives in kilograms).

The *hit probability* is the statistical likelihood of hitting a designated target. In actuality, this property of munitions may be quite complex (taking into account the deploying aircraft and target classes, for example); in this formulation, it is taken to be a fixed probability.

The *damage radius* is the maximum distance over which collateral damage may result (from blast or fragmentation effects).

5.10. Aircraft Schedules

An aircraft schedule is a set of *reservations,* each of which indicates a specific *action* to be performed over a specific *time period* (designated as a start time and an end time):

Reservation::

action:	AircraftAction
period:	TimePeriod

The following classes of actions are modeled: idling on ground, holding position airborne, repositioning, engaging a ground target, refueling, re-arming, electronic warfare support and defending strike aircraft. Further details of these actions follow.

GoundIdle

The aircraft lands (if necessary), remains on the ground and then takes off (if necessary).

Hold

The aircraft holds its current position. This action is included to account for the fuel consumed, e.g., by a refueling tanker waiting to rendezvous with a client aircraft.

Reposition::

destination:	Coordinates	(3-dimensional)
threats:	Set of ThreatAssessment	
support:	Set of Aircraft	

ThreatAssessment::

threat:	Target

level: Real

For repositioning, the destination (including altitude) needs to be specified. Repositioning actions may be annotated with information about threats from enemy defensive installations and support aircraft assigned to suppress those threats. For each threat, the installation is identified along with the level of threat (on some unspecified scale) that the installation represents.

EngageTargets::

targets:	Set of TargetEngagement	
threats:	Set of ThreatAssessment	
support:	Set of Aircraft	

TargetEngagement::

target:	Target	
attackAngle:	PlanarAngle	(optional)
munitions:	MunitionsClass → Natural	

For target engagements, a non-empty set of targets is specified. For each target, an optional attack angle may be specified, and the classes and quantities of munitions for attacking each target must be specified. Note that the engagement will take place at whatever position the aircraft has reached before executing this action: the schedule must be such that this position is near the targets, which may require the insertion of repositioning actions. In this problem formulation, any change in location that occurs during engagement is ignored.

Threat assessment and suppression may also be noted, as for repositioning.

GetFuel::

tanker:	Aircraft	
quantity:	FuelQuantity	(kilograms)

GiveFuel::

client:	Aircraft	
quantity:	FuelQuantity	(kilograms)

Refueling is modeled as two actions: one for the tanker providing the fuel and one for the aircraft receiving the fuel. The quantity of fuel required needs to specified (it is specified in both the tanker and client aircraft for convenience). As with target engagements, refueling actions do not contain implicit repositioning so the net effect of actions that are scheduled to occur before a refueling action must be such that the tanker and client aircraft are in (approximately) the same location. In this problem formulation, any change in location that occurs during refueling is ignored.

Rearm::

load: MunitionsLoad

For rearming, the classes and quantities of munitions need to be specified. Note that a rearming action specifies what the munitions load will be after rearming, rather than a quantity of munitions to be *added* to the aircraft's existing load.

The aircraft's schedule must be such that the aircraft will be at an appropriate location to rearm when this action is executed. Note that in this problem formulation, the facility where rearming takes place is not explicitly modeled: it is assumed to be capable of handling an arbitrary number of rearming actions simultaneously and has an infinite supply of each class of munitions.

StandoffEW::

targets:	Set of Target	
clients:	Set of Aircraft	

EscortEW::

destination:	Coordinates	(3-dimensional)
targets:	Set of Target	
clients:	Set of Aircraft	

Two types of electronic warfare are modeled: *standoff* and *escort*. In standoff EW, the EW aircraft remains at its current location and provides jamming support for the specified set of client aircraft. A set of known radars may be specified to be jammed; in addition, the EW aircraft may monitor the electromagnetic spectrum and jam whatever signals are detected.

In escort EW, the EW aircraft accompanies the client aircraft to the specified destination, providing jamming support along the way. (It may also deploy HARM missiles to destroy Red radars but in this problem formulation such actions are not explicitly scheduled.)

EscortDefense::

destination:	Coordinates	(3-dimensional)
targets:	Set of Target	
clients:	Set of Aircraft	

In an escort defense action, a fighter aircraft accompanies the specified client aircraft to the specified location and responds to Red threats as appropriate along the way, e.g., by destroying target acquisition radars or SAM sites. A set of known threats may be specified, but unknown threats may arise. The fighter aircraft should be appropriately pre-positioned to rendezvous with the client aircraft at the start of the defense action.

Note that only repositioning, escort EW and escort defense actions may result in significant changes in the aircraft's location.

5.11. Dynamic State of Aircraft

As an aircraft executes the actions in a schedule, certain properties – such as its position, fuel load and munitions load – change in manners that, in this problem formulation, are taken to be deterministic. Consequently, the *dynamic state* of an aircraft can be predicted from its schedule and its initial state.

In this problem formulation, an aircraft's dynamic state has the following properties:

AircraftDynamicState::

position:	Coordinates	(3-dimensional)
fuelLoad:	FuelQuantity	(kilograms)
refuelingLoad:	FuelQuantity	(kilograms)
munitionsLoad:	MunitionsLoad	

Although an aircraft can perform multiple simultaneous actions, such as flying and jamming, in this problem formulation simultaneous actions are combined into a single AircraftAction that must be completed before another can begin. Consequently, the AircraftActions for a single aircraft must not overlap.

Due to this constraint, a schedule's *set* of reservations can be considered to be a *sequence* of reservations that are ordered in time, with each action's scheduled end time being earlier than the scheduled start time of the following action (if there is one). For a given schedule, *reservation(i)* denotes the i^{th} reservation (for i between 1 and the number of reservations, inclusive).

It is useful in formulating the constraints below to consider the (predicted) effects of each action on an aircraft's dynamic state: given an initial dynamic state *D(0)*, the dynamic state after the first reservation, *D(1)*, can be computed based on the class and parameters of the action; then the dynamic state after the second reservation, *D(2)*, can be computed from *D(1)*; and so on.

If *D(i)* denotes the predicted dynamic state of an aircraft after the i^{th} reservation ($1 \leq i <$number of reservations), then *D(i+1)* can be computed from *D(i)* by considering the class of action specified by *reservation(i+1)*, as detailed below. For brevity, *position(i)* may be used to denote *position(D(i))*, and similarly for the other properties.

N.B.: if a property is not explicitly mentioned for a class of action, then it is assumed to be unchanged by that class.

- GroundIdle – This action is taken to be a *no-op* for aircraft, although in actuality landing and taking off would consume fuel.
- Holding – The fuel load is decreased by an amount depending on the duration of the holding action:
  fuelLoad(i+1) = fuelLoad(i) - burnModel(duration)
- Repositioning – The position of the aircraft after a repositioning action is the specified destination. The amount of fuel consumed depends on the distance traveled and the time taken, and is given by the aircraft's fuel burn model.
  position(i+1) = destination
  fuelLoad(i+1) = fuelLoad(i) - burnModel(distance, duration)
  where *distance* = |*destination* - *position(i)*|.
- Target Engagement – The quantity of each type of munitions is depleted by the number deployed against the targets: for each munitions class *m*:
  munitionsLoad(i+1)(m) = munitionsLoad(i)(m) - $\Sigma_{t \in targets}$ munitions(t)(m)
- GetFuel – The fuel load is increased by the specified amount:
  fuelLoad(i+1) = fuelLoad(i) + quantity
- GiveFuel – The fuel available for refueling other aircraft is depleted by the specified amount:
  refuelingLoad(i+1) = refuelingLoad(i) - quantity

- Rearm – The munitions load changes to the specified munitions load.
  munitionsLoad(i+1) = load
- StandoffEW – The fuel load is reduced by the amount of fuel required for the aircraft to hold position for the duration of the action:
  fuelLoad(i+1) = fuelLoad(i) - burnModel(duration)
- EscortEW – The position of the aircraft at the end of the action is the specified destination and the fuel load is reduced appropriately for the distance and duration:
  position(i+1) = destination
  fuelLoad(i+1) = fuelLoad(i) - burnModel(distance, duration)
- EscortDefense – The position of the aircraft after the action is the specified destination and the fuel load is reduced appropriately for the distance and duration:
  position(i+1) = destination
  fuelLoad(i+1) = fuelLoad(i) - burnModel(distance, duration)

Note that some of the dynamical properties may become *unphysical* by these computations, indicating that the schedule is *infeasible*. For example, the fuel load may become negative. The feasibility of a schedule in general is determined by the constraints detailed below.

5.12. Constraints on Aircraft Schedules

5.12.1. Contiguous Actions

As discussed above, no two actions for any single aircraft can be scheduled to overlap. Moreover, to help ensure that all resource consumption is accounted for, it is convenient to schedule all of an aircraft's time. Consequently, reservations are required to be contiguous – the end time of each reservation meets the start time of the following reservation (if any).

$$\forall\ i \in 1\ldots\text{size}(S)\text{-}1 \cdot \text{startTime}(\text{reservation}(i+1)) = \text{endTime}(\text{reservation}(i))$$

where *S* is an aircraft schedule, and *r* and *s* are reservations in that schedule.

In order to satisfy this constraint, it may be necessary to insert on-ground idling or airborne holding actions into a schedule.

5.12.2. Sufficient Fuel

This constraint specifies that the aircraft never runs out of fuel. Given the prediction of the aircraft's dynamic state, as discussed above, this constraint can be formulated in terms of the fuel load after each action:

$$\forall\ i \in 1\ldots\text{size}(S) \cdot \text{fuelLoad}(i)>0$$

5.12.3. Sufficient Duration

For repositioning, EscortEW and EscortDefense actions, the duration of the reservation must be long enough to allow the aircraft to travel the scheduled distance, given the aircraft's maximum speed.

For target engagement, refueling and rearming actions, the duration of the reservation must be long enough to allow the aircraft to complete the action. In this problem formulation, each of these classes of action is assumed to take a constant time.

For on-ground idling, sufficient time must be allowed at least for landing and take-off. (Moreover, it is probably not sensible to schedule very short on-ground idles.)

$\forall$ i$\in$1…size(S)·
if class(action(reservation(i))) = Reposition, EscortEW or EscortDefense:
  duration(reservation(i)) $\geq$ |position(i) - position(i-1)|/maximumSpeed
if class(action(reservation(i))) = GroundIdle:
  duration(reservation(i)) $\geq$ minOnGroundDuration
if class(action(reservation(i))) = EngageTargets:
  duration(reservation(i)) $\geq$ engagementDuration
if class(action(reservation(i))) = GiveFuel or GetFuel:
  duration(reservation(i)) $\geq$ refuelingDuration
if class(action(reservation(i))) = Rearm:
  duration(reservation(i)) $\geq$ rearmingDuration

5.12.4. Appropriate Landing

For on-ground idling and rearming, the aircraft must (approximately) be pre-positioned at an appropriate location (at an airbase, for example).

$\forall$ i$\in$1…size(S)· class(action(reservation(i))) = GroundIdle or Rearm
$\Rightarrow$ $\exists$L$\in$LandingSites· position(i) $\approx$ position(L)

where *LandingSites* is the set of suitable landing sites (taken to be a constant in this formulation).

5.12.5. Sufficient Munitions

For each target engagement action, the aircraft must have a sufficient load of each required class of munitions. Given the prediction of the aircraft's dynamic state, this condition is readily expressed in terms of the munitions load after each action:

$\forall$ i$\in$1…size(S), m$\in$MunitionsClasses· munitionsLoad(i)(m)$\geq$0

This constraint would be violated only by scheduling over-use of any class of munitions, causing the predicted load to become negative.

5.12.6. Kill Probability

For each target that is to be engaged, the combined probability of the deployed munitions destroying the target must exceed the target's minimum kill probability.

$\forall$ r$\in$S· class(action(r)) = EngageTargets
$\Rightarrow$ $\forall$ t$\in$targets(action(r))· combinedKillProbability(munitions(t), target(t))
  $\geq$ minKillProbability(target(t))

where the function *combinedKillProbability* computes the probability of the specified munitions killing the target, based on the munitions' individual effects models.

However, in this problem formulation, a simpler approach is taken: the *expected* weight of explosives deployed against a target must exceed the target's specified explosive weight, where the expected weight of explosives is the sum of the yields of the individual munitions scaled by the munitions accuracy:

$$\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{EngageTargets}$$
$$\Rightarrow \forall\ t \in \text{targets}(\text{action}(r)) \cdot \text{totalExplosiveYield}(\text{munitions}(t))$$
$$\geq \text{explosives}(\text{target}(t))$$

where *totalExplosiveYield* sums the yields of the individual munitions:

$$\text{totalExplosiveYield}(D\text{: MunitionsClass} \rightarrow \text{Natural})$$
$$= \Sigma_{m \in \text{MunitionsClasses}}\ D(m) \times \text{hitProbability}(m) \times \text{yield}(m)$$

5.12.7. At Most One Kill per Mobile Target

In this problem formulation, a mobile target should be scheduled to be engaged at most once because a mobile target's trajectory is likely to change significantly after the target is attacked. If multiple engagements are required, they should be scheduled one at a time, with successive engagements being informed by damage assessment.

$$\forall t \in \text{Targets} \cdot \text{isMobile}(t)$$
$$\Rightarrow |\{\ (a\text{:Aircraft}, r\text{:Reservation}) \cdot r \in \text{schedule}(a) \wedge \text{destroys}(r, t)\ \}| \leq 1$$

where *isMobile(t)* determines if the specified target is mobile (based on its trajectory) and *destroys(r, t)* determines if reservation *r* is a target engagement action against target *t*:

$$\text{destroys}(r, t) \equiv \text{class}(\text{action}(r)) = \text{EngageTargets}$$
$$\wedge\ \exists e \in \text{targets}(\text{action}(r)) \cdot t = \text{target}(e)$$

5.12.8. Proximity to Targets for Engagement

In order for an aircraft to engage a target, the aircraft must be close enough to the target at some point during the engagement, where "close enough" is defined by the stand-off distances of all of the munitions deployed against the target.

$$\forall\ i \in 1\ldots\text{size}(S) \cdot \text{class}(\text{action}(\text{reservation}(i))) = \text{EngageTargets}$$
$$\Rightarrow \forall\ t \in \text{targets},\ m \in \text{MunitionsClasses} \cdot \text{munitions}(t)(m) > 0$$
$$\Rightarrow \text{separation} \leq \text{standoffRange}(m)$$
$$\text{where separation} = \text{closestDistance}(\text{position}(i), \text{trajectory}(\text{target}(t)), \text{period}(\text{reservation}(i)))$$

The function *closestDistance* computes the smallest separation between the specified position and the specified trajectory that occurs during the specified time period.

5.12.9. Collateral Damage

For a target engagement action, the largest collateral damage range caused by any of the munitions deployed against the target must be smaller than the target's smallest collateral damage radius during the entire engagement period.

$$\forall\ i \in 1\ldots\text{size}(S) \cdot \text{class}(\text{action}(\text{reservation}(i))) = \text{EngageTargets}$$

$\Rightarrow \forall\ t \in targets(action(reservation(i))),\ m \in MunitionsClasses \cdot munitions(t)(m) > 0$
$\Rightarrow damageRadius(m) \leq damageLimit(target(t), period(reservation(i)))$

The function *damageLimit* computes the smallest value of the property *damageRadius* that occurs in the specified target's trajectory during the specified time period.

5.12.10. Angle of Attack

In this problem formulation, it is assumed that an aircraft can always adopt the requisite angle of attack.

5.12.11. Fuel Capacity

When an aircraft refuels, the quantity of fuel it receives added to the quantity it already has cannot exceed its fuel capacity. This constraint is readily expressible in terms of the dynamic state after refueling:

$$\forall\ i \in 1 \ldots size(S) \cdot class(action(reservation(i))) = GetFuel$$
$$\Rightarrow fuelLoad(i) \leq fuelCapacity$$

5.12.12. Refueling Load

A refueling tanker cannot provide more fuel than it has onboard. This constraint can be expressed in terms of the refueling load that remains after refueling actions:

$$\forall\ i \in 1 \ldots size(S) \cdot class(action(reservation(i))) = GiveFuel$$
$$\Rightarrow refuelingLoad(i) \geq 0$$

5.12.13. Refueling Rendezvous

In this problem formulation, a tanker can refuel only one client aircraft at a time. This constraint is implied by the Sequential Actions constraint since a single GiveFuel action can supply fuel to only one aircraft. However, it is necessary to ensure that both the tanker and client aircraft agree on refueling and that they are close enough to effect refueling.

For a schedule S_T of a refueling tanker T:

$$\forall\ i \in 1 \ldots size(S_T) \cdot class(action(reservation_T(i))) = GiveFuel$$
$$\Rightarrow \exists\ j \in 1 \ldots size(S_C) \cdot class(action(reservation_C(j))) = GetFuel$$
$$\wedge\ tanker(action(reservation_C(j))) = T$$
$$\wedge\ quantity(action(reservation_T(i))) = quantity(action(reservation_C(i)))$$
$$\wedge\ |period(reservation_T(i)) \cap period(reservation_C(j))| \geq refuelingDuration$$
$$\wedge\ |position_T(i-1) - position_C(j-1)| \leq refuelingProximity$$

where the client aircraft C is the one specified by the tanker's GiveFuel reservation – i.e., $C = client(action(reservation_T(i)))$ – and the subscripts T and C are used to distinguish properties of the tanker and client aircraft.

In words, this constraint can be read as follows: for a given reservation for a tanker to supply fuel, the specified client aircraft must have a corresponding reservation to get fuel from that tanker, and the two reservations must overlap in time sufficiently and the two aircraft must be sufficiently close at the start of the reservations.

5.12.14. Refueling Tanker Stays in Friendly Airspace

Air refueling tankers must remain in friendly airspace, defined by the set of coordinates *FriendlyAirspace*. For a schedule S_T of a refueling tanker T:

$$\forall\ i \in 1 \ldots \text{size}(S_T) \cdot \text{position}(i) \in \text{FriendlyAirspace}$$

5.12.15. Munitions Capacity

For rearming, the specified munitions load must be allowed by the munitions capacity of the aircraft, A:

$$\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{Rearm}$$
$$\Rightarrow \text{munitionsCapacity}(A)(\text{load}(r))$$

5.12.16. Threat Suppression

Repositioning and Target Engagement actions may be subject to threats from Red installations – it is assumed that a function *threatAssessment* is available that annotates the actions in an aircraft's schedule with threat assessments, based on the aircraft's trajectory.

Aircraft must be assigned to suppress threats, which they may achieve by: (i) standoff EW; (ii) escort EW; (iii) escort defense; or (iv) destroying the threatening installations before the threatened actions begin. (N.B.: the installations may already be scheduled to be destroyed as targets in their own right or because they are threatening earlier actions – it does not matter in this constraint *why* they are scheduled to be destroyed.)

The constraint considered here only checks that support aircraft have been assigned to threatened actions – the *effectiveness* of the assigned aircraft against the threats is considered in other constraints.

For a schedule S of an aircraft A:

$$
\begin{aligned}
&\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{Reposition or EngageTargets} \\
&\Rightarrow \forall\ t \in \text{threats}(\text{action}(r)) \cdot \\
&\qquad \exists\ b \in \text{support}(\text{action}(r)),\ s \in \text{schedule}(b) \cdot \\
&\qquad\qquad \text{standoffEW}(s, \text{threat}(t)) \wedge \text{period}(s) \subseteq \text{period}(r) \\
&\qquad \vee \exists\ b \in \text{support}(\text{action}(r)),\ s \in \text{schedule}(b) \cdot \\
&\qquad\qquad \text{escortEW}(s, \text{threat}(t)) \wedge \text{period}(s) \subseteq \text{period}(r) \\
&\qquad \vee \exists\ b \in \text{support}(\text{action}(r)),\ s \in \text{schedule}(b) \cdot \text{defends}(s) \wedge \text{period}(s) \subseteq \text{period}(r) \\
&\qquad \vee \exists\ b \in \text{Aircraft},\ s \in \text{schedule}(b) \cdot \text{destroys}(s, \text{threat}(t)) \wedge \text{period}(s) \leq \text{period}(r)
\end{aligned}
$$

where $p \subseteq q$ signifies that time period p includes time period q, $p \leq q$ signifies that time period p finishes before time period q begins, and:

- *standoffEW(s, t)* determines if reservation s is a standoff jamming action against target t for aircraft A:

$$\text{standoffEW}(s, t) \equiv \text{class}(\text{action}(s)) = \text{StandoffEW}$$
$$\wedge\ A \in \text{clients}(\text{action}(s)) \wedge t \in \text{targets}(\text{action}(s))$$

- *escortEW(s, t)* determines if reservation *s* is an escort EW action against target *t* for aircraft *A*:

$$\text{escortEW}(s, t) \equiv \text{class}(\text{action}(s)) = \text{EscortEW}$$
$$\wedge\ A \in \text{clients}(\text{action}(s)) \wedge t \in \text{targets}(\text{action}(s))$$

- *defends(s)* determines if reservations *s* is a defense action for aircraft *A:*

$$\text{defends}(s) \equiv \text{class}(\text{action}(s)) = \text{Defend} \wedge A \in \text{clients}(\text{action}(s))$$

- *destroys(s, t)* determines if reservation *s* is a target engagement action against target *t*:

$$\text{destroys}(s, t) \equiv \text{class}(\text{action}(s)) = \text{Defend} \wedge A \in \text{clients}(\text{action}(s))$$

5.12.17. Standoff EW Proximity

To effectively shield a client aircraft against a threat from a standoff position, an EW aircraft, *E*, must be sufficiently close to both. In this problem formulation, this constraint is expressed in terms of maximum separations – between the EW and client aircraft and between the EW aircraft and the threats – that must be observed throughout the duration of the EW action.

$$\begin{aligned}
&\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{StandoffEW} \\
&\Rightarrow \wedge\ \forall\ C \in \text{clients}(\text{action}(r)) \cdot \text{maximumSeparation}(\text{trajectory}(E), \\
&\qquad\qquad\qquad\qquad \text{trajectory}(C), \text{period}(r)) \\
&\qquad\qquad \leq \text{standoffEWRange} \\
&\wedge\ \forall\ R \in \text{targets}(\text{action}(r)) \cdot \text{maximumSeparation}(\text{trajectory}(E), \\
&\qquad\qquad\qquad\qquad \text{trajectory}(R), \text{period}(r)) \\
&\qquad\qquad \leq \text{standoffEWRange}
\end{aligned}$$

where the function *maximumSeparation* computes the largest instantaneous distance between the two specified trajectories during the specified time period.

5.12.18. Escort EW Proximity

To effectively escort a client aircraft, an EW aircraft, *E*, must remain sufficiently close to the client.

$$\begin{aligned}
&\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{EscortEW} \\
&\Rightarrow \forall\ C \in \text{clients}(\text{action}(r)) \cdot \text{maximumSeparation}(\text{trajectory}(E), \\
&\qquad\qquad\qquad\qquad \text{trajectory}(C), \text{period}(r)) \\
&\qquad\qquad \leq \text{escortEWRange}
\end{aligned}$$

N.B.: in actuality, an EW aircraft may also wish to maintain a minimum distance from the threats for self-protection. However, it is assumed that this requirement can be accommodated within the confines laid down by the escort action.

5.12.19. Standoff EW Capacity

The number of threats against which an EW aircraft, *E*, can provide protection from a standoff position is limited. (The condition that the capacity be non-zero ensures that only EW-capable aircraft are scheduled to provide EW support.)

$$
\begin{aligned}
&\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{StandoffEW} \\
&\Rightarrow \text{EWCapacity} > 0 \\
&\wedge\ |\text{threats}(\text{action}(r))| \leq \text{EWCapacity}(E)
\end{aligned}
$$

5.12.20. Escort EW Capacity

The number of simultaneous threats against which an EW aircraft, *E*, can provide protection during escort is limited. (The condition that the capacity be non-zero ensures that only EW-capable aircraft are scheduled to provide EW support.)

$$
\begin{aligned}
&\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{EscortEW} \\
&\Rightarrow \text{EWCapacity} > 0 \\
&\wedge\ \forall\ p \in \text{period}(r) \cdot |\{t \in \text{threats}(\text{action}(r)) \wedge \text{separation}(\text{trajectory}(E), \text{trajectory}(t), p) \\
&\qquad\qquad\qquad\qquad < \text{airThreatRange}(t)\}| \\
&\qquad\qquad\quad \leq \text{EWCapacity}(E)
\end{aligned}
$$

where *separation* computes the instantaneous distance between the two specified trajectories at the specified time.

5.12.21. Escort Defense Proximity

In order to effectively defend its client aircraft, a support aircraft must remain within a certain distance of its clients. For a schedule *S* of a defending aircraft *F*:

$$
\begin{aligned}
&\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{EscortDefense} \\
&\Rightarrow \forall\ C \in \text{clients}(\text{action}(r)) \cdot \text{maximumSeparation}(\text{trajectory}(F), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{trajectory}(C), \text{period}(r)) \\
&\qquad\qquad\qquad\qquad \leq \text{defenseRange}(F)
\end{aligned}
$$

5.12.22. Escort Defense Capacity

If a support aircraft, *F*, is assigned to defend strike aircraft against known threats, the number of simultaneous threats must be limited. Note that only known threats are taken into account.

$$
\begin{aligned}
&\forall\ r \in S \cdot \text{class}(\text{action}(r)) = \text{EscortDefense} \\
&\Rightarrow \text{class}(F) = \text{Fighter} \\
&\wedge\ \forall\ p \in \text{period}(r) \cdot |\{t \in \text{threats}(\text{action}(r)) \wedge \text{separation}(\text{trajectory}(F), \text{trajectory}(t), p) \\
&\qquad\qquad\qquad\qquad < \text{airThreatRange}(t)\}| \\
&\qquad\qquad\quad \leq \text{defenseCapacity}(F)
\end{aligned}
$$

where *separation* computes the instantaneous distance between the two specified trajectories at the specified time.

N.B.: when a support aircraft is scheduled to perform defense actions, the number of threats to be countered may not be known in advance – some threats may be specified but other, unanticipated threats may arise. While the quantity of munitions required is thus unknown, some account should be taken of how many defense missions are scheduled between re-armings. However, this is omitted from this problem formulation.

5.12.23. Quality and Cost Metrics

Given an aircraft schedule *S*, its quality can be computed as the sum of the values of the targets engaged, weighted by their probabilities of being killed:

$$\text{targetValue}(S) \equiv \Sigma_{r\in S \wedge \text{class}(\text{action}(r))=\text{EngageTargets}} \Sigma_{t\in\text{targets}(\text{action}(r))} \text{value}(\text{target}(t)) \times \text{combinedKillProbability}(\text{munitions}(t), \text{target}(t))$$

The cost of a schedule can be measured in terms of the fuel and munitions consumed. The fuel cost can be computed over all refueling actions (with allowance made for the initial and final fuel loads):

$$\text{fuelCost}(S) \equiv \Sigma_{r\in S \wedge \text{class}(\text{action}(r))=\text{GetFuel}} \text{quantity}(\text{action}(r)) + \text{fuelLoad}(0) - \text{fuelLoad}(\text{size}(S))$$

The munitions cost can be computed over all target engagements, over all targets engaged in each engagement, and over all munitions deployed against each target:

$$\text{munitionsCost}(S) = \Sigma_{r\in S \wedge \text{class}(\text{action}(r))=\text{EngageTargets}} \Sigma_{t\in\text{targets}(\text{action}(r))} \Sigma_{m\in\text{MunitionsClasses}} \text{unitCost}(m) \times \text{munitions}(t)(m)$$

For a set of aircraft, quality and cost metrics can be computed as the sums of the individual metrics.

6. Planware

The Planware system is Kestrel's domain-specific generator of high-performance planners and schedulers.

Planware provides an answer to the question of how to help automate the acquisition of requirements from the user, and how to assemble a formal requirement specification for the user. The key idea is to focus on a narrow, well-defined class of problems and programs, and to build a precise, abstract, domain-specific description formalism that covers this class.

From a software development perspective, interaction with the user is only required in order to obtain the refinement from the abstract specification to a description of the concrete requirements of a particular problem instance.

Planware's main capabilities are:

Simple Modeling Language: A simple, intuitive formalism that describes the behavior of a resource by explicitly describing the set of activities it must perform.

Multiple Resource Coordination: The synchronized use of multiple resources is obtained by the use of compact descriptions of resources capabilities represented as services.

Configurable Problem-Solving Strategy: The modeler can select the scheduling strategy to be used by the generated scheduling application just by choosing among a number of different search-based implementations available to the code generator.

Integrated Development Environment: Modeling, scheduler generation, and schedule computation are performed in a uniform development environment that supports all phases of the development process.

6.1. Planware in a Nutshell

One of the most important requirements driving the design and implementation of Planware was the ability to represent the synchronization of multiple resources without using complex mathematical or logical formalisms. To achieve this goal, Planware uses Abstract State Machines (ASM) to model the behavior of tasks and resources.

To handle the interactions involved in multi-resource problems, Planware uses a service matching theory in which resources can offer and/or require different types of services. The scheduler is responsible for matching providers and requesters in a consistent fashion. For example, a transportation organization might want a scheduler to simultaneously handle its aircraft, crews, fuel, and airport load/unload facilities. Each resource has its own internal required patterns of behavior and may have dependencies on other resources.

The semantics of a resource is the set of possible behaviors that it can exhibit. We treat these behaviors as (temporal) sequences of activities modeled as ASM modes (abstract states). Each activity is characterized by a set of mode variables (e.g., start-time and duration), the set of services that it offers (e.g. the flying mode of an aircraft offers a transportation service), and the set of services that it requires (e.g. the flying mode of an aircraft requires the services of a crew).

A formal theory of a resource should have as models exactly the physically feasible behaviors of the resource. The axioms serve to constrain the values it can exhibit. A formal theory constrains the values that mode variables can take on in states (e.g., the weight of cargo cannot exceed a maximum value during the flying mode of an aircraft). The transitions serve to constrain the evolution of the mode variables (e.g., the finish time of one activity must occur no later than the start time of the next activity; a take-off activity can only be followed by a fly activity, etc.).

All modes have variables for the start-time, finish-time, and duration of the corresponding activity, plus related constraints. They may also have other variables and constraints that further define the resource behavior, and better describe the mode. A mode may also provide and/or require services. Only the temporal resource constraints of the activities are relevant, not their nature. Thus the same mechanism can be used to model activities as diverse as transportation, computation, allocating a logical design to a hardware/software platform, personnel assignments, workflow, manufacturing, and so on.

A task is also expressed formally as an ASM. The main difference between a task and a resource is that a task offers no service – it only requires services of resources. For example, a fighter mission may require fuel, crew, and weapons resources.

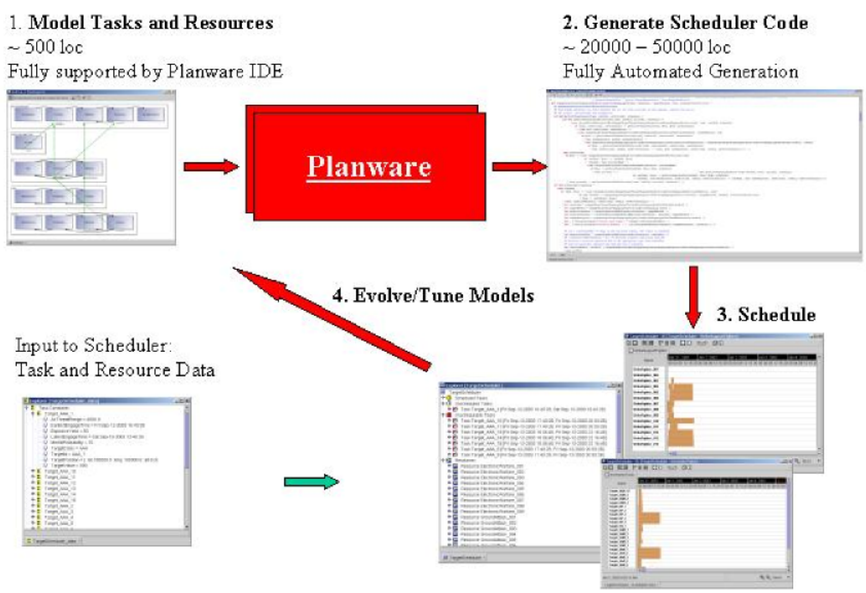

Figure 12:Planware Generation Process

6.2. Representing Resources as Activity Machines

To allow users to specify complex multi-resource problems Kestrel started by trying to identify a language that would naturally represent the basic concepts in the domain. The first approximation was to look for planning and scheduling ontologies.

Smith & Becker in [19] describe an ontology for planning and scheduling systems. The five top-level entities in this ontology are tasks or demands, activities, resources, services, and constraints. Using this ontology, the role of a scheduling or planning system can be described as the prescription of a sequence of activities that a set of resources must perform over time to perform the services required by a task. Based on this description, it is clear that any formalism for describing a scheduling problem domain must be able to represent tasks, resources, activities, and services, plus associated constraints.

If one considers the processing of an activity by a resource as a possible "state" or "mode" the resource entity can assume, one can think of a resource model as the description of all the valid sequences of activities it can perform. For example, a transportation aircraft might have the following sequence of activities:

Prepare → Fly → Fly → Unload → Refuel → Fly → and so on.

Or a strike fighter might execute the following sequence of activities while executing a mission:

Rearm → Position → EngageTarget → Position → EngageTarget → and so on.
Each legal sequence of activities is called a behavior. A resource is characterized by a potentially infinite collection of behaviors. A convenient and intuitive model for concisely representing an infinite collection of behaviors is a state machine but since the term "state" is somewhat misleading the models are referred to as activity machines, as exemplified below.

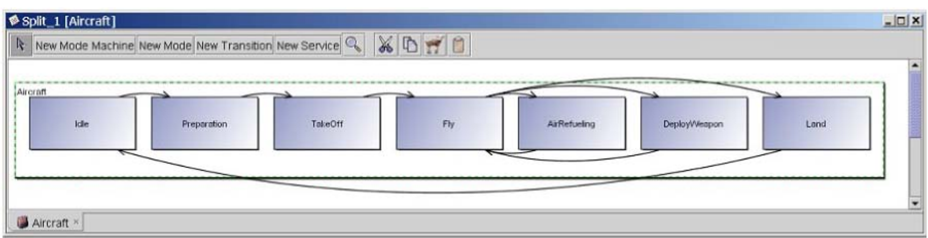

Figure 13:A Planware activity machine diagram

In the activity machine diagram, the boxes/states are referred to as activities. The arrows are called transitions and indicate which activities can legally follow one another.

6.3. Representing Activities as Sets of State Variables

The activity machine *per se* gives general information about the legal sequences of activities for a resource, but little information about the activities. Clearly in scheduling there is a need to model the timing of activities, as well as the capacity of a resource to do useful work, and other physical constraints. The next step in modeling resources is to represent information about an activity via activity variables, simply called variables.

The set of variables defined for a particular resource model represents the state descriptor used to describe all activities performed by this resource. For example, a Fly activity of a transportation resource might be characterized by variables that model its start time, finish time, origin, destination, and others. Planware uses the same collection of variables to characterize all activities of a particular activity machine. This collection is sometimes called the signature of the machine. Technically, each activity is treated as a first-order theory, with the signature providing the vocabulary of the theory, and the axioms specifying constraints on the meaning of the variables (and other vocabulary).

```
mode-machine StrikeFighter is
    constant baseLoc                                    : Location
    constant fuelBurnRate                               : BurnRate
    constant maxFuelCapacity                            : Capacity
    constant engagementDuration                         : Duration
    constant rearmingDuration                           : Duration
    input-variable targetLoc                            : Location
    input-variable munitionType                         : Munition
    input-variable targetYield                          : Capacity
    internal-variable origin, dest, airRefTrack         : Location
    external-variable st, et, duration                  : Time
end-mode-machine
```

Figure 14:Activity Machine Signature for a Strike Fighter Resource

The variables defining the signature of an activity machine are further divided into four groups: constants, internal variables, external variables, and input variables.

Constants: are the fixed parameters used to characterize invariant aspects of the resource. For example, the size of the cargo hold of an aircraft, its maximum speed, the maximum amount of fuel it can carry are parameters that can be constants for a given aircraft type. The set of constants define the input values that should be provided to the scheduling system to create concrete instances of resources at schedule computation time.

Internal Variables are auxiliary variables used by the scheduler for bookkeeping purposes. For example, if there is a constraint that states that an aircraft needs to go through preventive maintenance after a certain number of hours flown, an internal variable can be used to maintain the number of hours flown since last maintenance.

External Variables represent the values that are modified by the scheduler engine while computing the schedule. For example, the start and end time of an aircraft Fly activity will be set by the scheduler based on the availability of additional resources like Crews and

Airports. External variable values depend not only upon the evolution of the resource machine but also upon the scheduling decisions made by the problem-solving control strategy.

Input variables Input variables serve as a mechanism for passing parameters between a resource requesting a service and a resource providing it. For example, the origin and destination of a Fly activity may be specified as input variables whose values get assigned each time a transportation task needs to be satisfied. Input variables act as constant variables for a given fragment of the resource behavior. The role of the input variables and external variables will be further discussed when we present the concept of services and service matches.

After defining the variables that can take on possibly new values at each activity, we can express more information about a behavior.

Most variables of interest hold their values for the duration of an activity, and only change with the transition between activities. In the syntax construct used to describe valid transitions, the variables whose values change as a result of the transition are explicitly represented. Three external variables are pre-defined for all activity machines, and do not need to be explicitly represented in the assignments field of a transition: start-time, end-time, and duration. Implicit in the structure of the activity machine is the constraint that states that the start-time of an activity is always greater than or equal to the end-time of the preceding activity; and that the end-time of an activity is equal to or greater than the sum of its start-time and its duration. Additional properties about the possible values a variable can take can be stated through the use of constraint expressions.

6.4. Defining Behavior using Constraints

Not all combinations of values for the variables are physically possible. For example, if the maximum munitions capacity of a fighter is 100 tons, then an activity in which the munitionRequired variable has a value exceeding 100 tons does not model a realizable situation. To rule out such impossible situations, each activity has axioms that express constraints on the values that variables can take on in an activity. The generated scheduler uses the constraints expressed in the model as pruning conditions to drive the expansion of the search tree.

Furthermore, it is necessary to put constraints on transitions, to model the physically realizable evolution of variables between activities. For example, the transition from *Preparation* to *TakeOff* specifies that the end time of activity *Preparation* should be less than or equal to the start time of activity *TakeOff*. The constraints labeling a transition must refer to the values of variables in both the before and after modes. The usual notation is to refer to the value of a variable **x** in the after state by priming it: **x'**. So the constraint **endTime' <= startTime** means that the finish time of the before activity must be no later than the start of the after activity. As previously mentioned, a number of temporal constraints between modes do not need to be explicitly represented since the structure of the machine already assumes temporal dependencies between the sequence of valid activities. The constraints on the transitions are called guards. Guards are used to guide the scheduler to expand the correct sequence of activities needed to satisfy the request.

Similar to the guards on the transition, the assignment of values to variables can also be used to express or enforce constraints. Conditional expressions can be used in the definition of variable assignments to drive the valid sequence of activities expanded. The following figure shows an example of a simplified Planware model for a strike fighter with four activities: Idle, Rearming, Positioning, Refueling, and EngagingTarget, and some of the valid transitions between the different modes. In the transition from Rearming to Positioning we determine if a Refueling activity is needed or not: If there is enough fuel to engage the target and return to the home base, no refueling is needed; otherwise, the value of the variable airRefTrack is set to the location of the air refueling track. The variable airRefTrack is used in the guard of the transition from Positioning to Refueling to force the scheduling algorithm to include an air refueling activity in the activity sequence if necessary.

```
mode-machine StrikeFighter is
  mode Idle has
  end-mode
  mode Rearming has
  required-invariant loadMunition (st, et, rearmingDuration, munitionType)
end-mode
mode Positioning has
end-mode
mode EngagingTarget has
  provided-invariant engageTarget(st, et, targetLoc, munitionType, targetYield)
end-mode
transition from Idle to Rearming when {} is
{ duration := rearmingDuration }
transition from Rearming to Positioning when {} is {
          dest := (if  (consumedFuel(baseLoc, targetLoc, fuelBurnRate) <= maxFuelCapacity)
                then findAirTrackLocation(baseLoc, targetLoc)
    else targetLoc),
    airRefTrack := (if (consumedFuel(baseLoc, targetLoc, fuelBurnRate) <= maxFuelCapacity)
   then dest
              else zeroLoc)}
        transition from Positioning to Refueling when { airRefTrack != zeroLoc } is {}

        transition from Positioning to Engaging when { dest = targetLoc } is {}

end-mode-machine
```

Figure 15:Activity Machine for Strike Fighter Resource

6.5. Coordinating Resources Using Services

The modes or activities, the variables, the transitions, and the constraints are sufficient to represent the behavior of an individual resource. The key missing element of this formalism is how to connect resources to tasks, and how to coordinate the usage of several resources to accomplish complex tasks. For example, the transportation of a certain amount of cargo between two locations may involve the usage of a number of different aircraft, airports, crews, fuel, ground control personnel, diplomatic clearances, etc. Engaging a given target may involve the coordination of air refueling tankers, escort aircraft, air patrol, jammers, etc.

We need to provide modeling constructs that allow the explicit representation of these dependencies. The missing modeling construct is the service: The service is the element used to coordinate and synchronize the execution of activities across different resources. Each activity machine may specify required and/or provided services. Machines that only request services define the top-level tasks that drive the entire scheduling process. Resources are machines that provide one or more services. Resources can also request additional services. For example, to provide transportation service to a transportation request, the aircraft may need services from one or more crew resources. The figure above gives an example of a resource that offers an *engageTarget* service and requires a *loadMunition* service to be able to engage the target. In this case, the resource plays the dual role of provider and requester. The concept of requested and provided services allows tasks and resources to be represented using a uniform formalism.

As illustrated above, a service is specified by a predicate associated with a mode together with an indication of whether it is a provided or required service. For example, the engage target service may be represented by the predicate *engageTarget(startTime, endTime, targetLoc, targetMunitionType)* which specifies a certain target located at coordinates specified as *targetLoc* to be engaged some time during the time interval defined by the values of *startTime* and *endTime*.

The requester resource specifies the service as a required condition. The provider specifies it as a provided condition. For temporal synchronization, a service can be specified as a pre-condition, a post-condition, or an invariant. If the service is specified as an invariant, both activities, the requesting and the providing, should start and end at the same time. For the other types, there are set of rules to establish the appropriate synchronization depending on the characteristics of the provider and requester. For example, if the requesting service is a pre-condition and the providing service is a post-condition, the providing activity should finish before the requesting activity can start.

6.6. Passing Parameters through Service Descriptions

There is also a set of rules governing the assignment of values to the parameters specified in the service description. For the requesting resource, external variables present in the service predicate will have their values set by the scheduler after an appropriate provider has been identified. All other variables will not change value. For the providing resource, input variables present in the service predicate will act as constants for the purpose of finding a valid sequence of activities to satisfy the request. External variables will be unified with external variables coming from the requester. At scheduling time, any constraints imposed on the external variables of the requester, will be translated to the corresponding variables of the provider. In our *engageTarget* example, the variables *targetLoc* and *targetMunitionType* are defined as input variables. Their values will be passed by the requesting target and will be treated as constant for the duration of that particular mission.

Note that the introduction of services in mode machines allows one to treat tasks as a special case of resource. Tasks are the drivers of a planning or scheduling problem. The overall nature of the scheduling problem is to carry out a set of tasks subject to the constraints imposed by the available resources. In terms of the activity machine model, a task can be modeled as a resource that requires service, but offers none. The figure below shows an air strike task modeled as a simple machine with just one mode. Its *engagingTarget* mode requires the service *engageTarget*. The task model also specifies the values to be used in the service request: target position and *munition* type are defined as constant parameters and should be provided as input to the provider resource. Note that the *engageTarget* mode has two axioms expressing constraints on the start and finish time of the activity – the start time must occur no earlier than the *earliestTimeOnTarget* and the finish time must occur before the *latestTimeOnTarget*.

```
mode-machine Target is
constant targetClass : TargetClass
constant minKillProbability : Probability
constant targetLoc : Location
constant earliestTimeOnTarget, latestTimeOnTarget : Time
constant munitionType : Munition
external-variable st, et, duration : Time
mode EngagingTarget has
   required-invariant engageTarget(st, et, targetLoc, munitionType)
   constraint st >= earliestTimeOnTarget
   constraint et <= latestTimeOnTarget
end-mode
end-mode-machine
```

Figure 16:Target Task Model

The component tasks and resources have been modeled individually, and now the composite system must be modeled. To model a complex resource system one focuses on the interactions of the components, which are specified by the services. The service match formula schema below expresses the conditions under which the service provided by resource Prov satisfies the service required by resource Req:

□ (constants (Req), input-vars (Req), constants (Prov))
□ (ext-vars (Req), internal-vars (Req), input-vars (Prov), ext-vars (Prov), internal-vars (Prov)).
(Provided Conditions (Req) ∧ ProvidedConditions(Prov)
⇒
ReqConditions(Req) ∧ Constraints (Req) ∧ ReqConditions(Prov) ∧ Constraints (Prov))

Two kinds of information are derived from reasoning about the formula. First, witnesses are obtained for the existentials, meaning that for each existentially quantified variable, a term over the preceding universally quantified variables is extracted. Second, any of the conjuncts in the consequent of the formula that cannot be proved are gathered. These gathered constraints are the aggregated constraints of the composite resource Req - Prov.

While they are not provable at design time, they will be treated as constraints to enforce at run-time (i.e. schedule-computation-time), either via pruning or constraint propagation.

In Planware the actual ground constraints are determined dynamically, and depend on the input data, together with the dynamics of the scheduling process (the current state of the process). This is in contrast to many Operations Research and Constraint Programming systems in which a static set of constraints is passed to a generic solver. Planware not only generates a customized solver for each problem, but the solver works on a dynamic constraint problem.

6.7. Scheduler Code Generation

From the activity machine models described in the previous section, Planware automatically generates a fully operational scheduling application. The key component of the generated code is a search-based scheduling algorithm, and a constraint propagation mechanism.

In addition to the search algorithm implementation, support code is also generated to represent resources and activities, and to produce I/O for the application. The following paragraphs explain in more detail the code generation process, and the different components created by Planware.

6.8. Scheduling Implemented as a Bidding Process

Planware generates search-based scheduling algorithms implementing a bidding process as its main control cycle. In this process, entities requiring services post tasks, or requests for bids. Provider resources capable of performing the type of service specified in a task respond with their best bid according to their own internal strategy. The requesters then collect the bids, rank them according to the requester's objective function, select the best bid, and notify the selected bidders. Constraint propagation is triggered every time a bid is accepted. The propagation updates the internal state of the resources involved in the bidding. The rejected bids are discarded, and no additional work is needed.

6.9. Algorithms Generated by Composing Program Schemas

The concrete implementation of the scheduling algorithm used in a particular application is obtained by instantiating and composing program schemes. A program schema is a parameterized fragment of algorithmic logic that gets instantiated for each service match between a given provider and requester. Different program schemes are used to allow a number of different bidding generation and bidding selection mechanisms to be combined in the implementation of an application. For example, a program schema could be used to implement a bidding mechanism in which the first feasible bid is accepted; a different one could collect up to n bids, and select the one that can finish the service with the minimum amount of time; a third one could generate all possible bids and select the one with earliest start-time.

6.10. Schema Composition Driven by Service Tree

The program schemes are composed in a tree-like structure reflecting the structure of service matches defined by the activity machine models. As described in the previous section, an activity machine can request, and/or provide services. For each required service in the model, the generator code will search for a matching provider – a resource providing a service that matches the signature of the required service. As a provider for a given service can request additional services from other resources, the service matches define a directed acyclic graph we refer to as the service-match tree. The code generator traverses this service tree creating the appropriate code to formulate the task, generate the bids, and select the best bid.

6.11. Scheduling Strategy Selected through Service Match

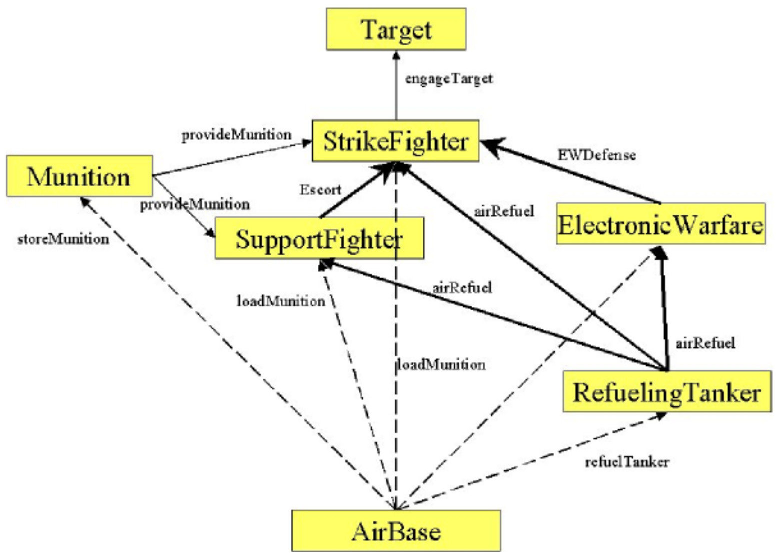

Figure 17:Service Tree for Target Scheduler

The service-match tree is an auxiliary data-structure automatically generated by Planware before the actual application generation. By exposing the structure of the service-match tree to the user through the graphical interface, a greater level of control over the code generation can be obtained. An advanced user can configure the search schemes to be used by the generator for each service match in the tree. The above figure shows a possible service tree for a model in which, to engage a target, Strike Fighter may require escort, EW defense, air refueling, and munition. Each of the required resources may also require additional resources. In this example, the munition resource requires some storage space at the AirBase.

Through the GUI, the user can select the search strategy used to satisfy the requests, as well as the sequence in which the services are satisfied. Once a provider and a requester for a given service are specified, the system automatically generates and inserts in the model source file a textual representation describing the service tree containing all possible matches between the different resources. The figure below shows the service match between the target task and the strike fighter. In this example, there is only one type of resource that can provide the engageTarget service. In general, several different resources could offer the same service.

6.12. Constraint Propagation Implemented by Arc Consistency Algorithm

The constraint propagation code used to update the resources is not automatically synthesized. A standard implementation of an arc consistency algorithm that propagates temporal constraints on a simple temporal network is used. All schedulers generated share the same implementation.

```
service-match Target EngagingTarget engageTarget is
service-requester Target
requester-mode EngagingTarget
requested-service required-invariant engageTarget
search-strategy BreadthFirst
service-providers has {
    provider-record StrikeFighter Engaging engageTarget is
    service-provider StrikeFighter
    provider-mode Engaging
    provided-service
        provided-invariant engageTarget
    search-strategy BreadthFirst
    end-provider-record
    }
end-service-match
```

Figure 18:Service Match generated for Target task and StrikeFighter

The constraint propagation is responsible for maintaining consistent start times for all scheduled activities. Each activity has a time bound representing the earliest and latest time the activity can start executing. Each activity time bound defines a node in the constraint network. The scheduler adds temporal constraints (arcs) between time bounds (nodes) as the problem solving process evolves. If constraint violations are detected, scheduling decisions are retracted, and the search backtracks to the last decision point before the violation.

6.13. Resource Represented as Capacity Profile

Activities and resources are closely related. Resources are represented by a capacity profile: a temporal sequence of activities representing the resource reservations performed by the scheduler. The profile represents a trace of the activity machine defined in the abstract model. The data structure used to represent the profile must be optimized for lookup and update. During the bid creation phase of the scheduling algorithm, the providers inspect their capacity profile searching for feasible intervals capable of feasibly performing the requested service. Once the requester accepts a bid, the selected provider updates its own profile to reflect the new reservation. Planware uses a binary tree implementation optimized for the particular type of resource.

The representation of the activities in the resource profile is generated from the set of variables defined by the activity machine model. All activities created for a given resource instance share the same set of constants. Activities are defined as a record structure with a field for each variable described in the model. The code to access and set each one of these fields is automatically generated. Additional code to print and display individual activities, and activities sequences is also generated to facilitate debugging, testing, and schedule visualization. A number of different output formats are supported: plain text, XML, etc.

6.14. Activity Sequences computed Dynamically

Activities are dynamically created at schedule computation time. Activity sequences are created by the bidding mechanism previously discussed. The generation of the bid creation mechanism is one of the most complex components of Planware. It involves the generation of code capable of querying the resource profile for feasible intervals, expanding the sequence of activities the resource must execute, and enforcing the constraints imposed on the service by both the requester and the provider resource.

The bid creation mechanism is implemented as a 3-step process: Identify feasible capacity intervals, expand activity sequence for selected intervals, propagate temporal constraints. If these three steps generate a feasible activity sequence, a bid containing this sequence is sent to the tasker resource. If the bid is accepted, then the resource profile is updated to include the new sequence of activities.

After a bid has been accepted, and the resources appropriately updated, the search algorithm changes its focus to schedule additional pending service requirements. Depending on the configuration provided by the user through the service match, the search proceeds by scheduling the requirements of the current bidder, or goes back to the level of the previous requester, and schedules its next task. The sequence of services scheduled is also determined by the service match structure.

In terms of the global behavior of the application, the execution of the generated scheduler starts by reading a file describing all the top-level tasks, and all concrete resources available. The scheduling algorithm cycles through the top-level tasks, and expands the search following the structure defined by the service-match tree. If a top-level task cannot be satisfied using the available resources, it is marked “unschedulable” and discarded. Once there are no more pending tasks in the system, the scheduling algorithm finishes its execution and, if instructed to do so, outputs the schedule in text form, writes the schedule to an XML file, and/or displays the schedule on the GUI.

6.15. Planware Implementation – IDE for Domain Modeling and Planning

Planware's implementation can be divided into two main components: the graphical interface, and the code generator.

6.15.1. Planware Interface Implemented as a NetBeans Module

The interface component is written in Java and uses the open source configurable Integrated Development Environment (IDE) platform NetBeans, which is an extensible IDE designed to support multiple programming languages and formalisms. Additional capabilities are added to the NetBeans platform by writing modules using its customization API. A NetBeans module is just a JAR file (collection of compressed Java class files) that can be "installed" in the platform. A module can implement a number of different capabilities like syntax sensitive source code editing, compilation, execution, and debugging among others. The Planware module provides:

1. An outline editor for editing activity machines based on a hierarchical representation of the models.
2. A graphical editor that allows the visualization and fast specification of activity machines.
3. A source code editor for more detailed specification of the models.
4. Visualization tools for inspecting the results of executing the generated code on test data.

Developing resource models in this environment requires very little knowledge of Planware syntax. The set of syntax constructs is small and most of the model creation activity can be accomplished using just the outline and graphical editors.

A typical Planware resource model has less than a hundred lines of code, and can be created in a matter of minutes. A complete application model can be defined in few hours using a highly interactive environment.

The advantage of using an extensible platform like NetBeans is that the full application development and execution can take place in the same environment, and using the same interaction paradigm. Defining models, generating and compiling code, and executing the scheduler are all defined using the same basic set of actions and gestures.

6.15.2. Planware Code Generator Implemented as a Specware Application

The Planware code generator is implemented as an application layer on top of Specware, Kestrel's software synthesis platform. The Planware code generator translates the activity machines and service match structure into an implementation of the algorithms and auxiliary data-structures described above. Planware first generates an intermediate representation of the algorithms in MetaSlang, the specification language used internally by Specware. This representation is then further refined, optimized, and composed with appropriate library code to generate a highly optimized implementation of the scheduling application in some programming language.

For a problem model with five different resource types (approximately 500 lines of code), Planware generates an intermediate representation with around 10,000 lines of code. The size of final code in the target language usually increases by a factor of 3 or 4 in comparison with the generated code since all the library code used is included as part of the target implementation. The total synthesis time is on the order of 1 or 2 minutes for average size models – 4 or 5 different types of resources.

In terms of run-time performance of the generated schedulers, without any special heuristics added, models with four resource types running on data sets with thousands of tasks, and around 20 resource instances for each resource type, generate schedules in a matter of seconds. The runtime performance of the generated code was around 20% faster than the performance provided by scheduling applications previously developed manually by the authors for the domain of logistical deployment.

6.16. Planware Application Development Process

The Planware domain analysis and application development process has the following steps:

Requirement Acquisition – The user interactively develops a model of the scheduling problem using the primitives previously discussed. This model describes the kinds of tasks and resources that are of concern. The figure below shows the graphical representation of a problem model.

The problem model is formalized into a specification that can be read abstractly as follows: Given a collection of task instances and a collection of resource instances, find a schedule that accomplishes as many of the tasks as possible (or approximately optimizes the given cost function), subject to all the constraints of the resource models, and using only the given resources.

The required and offered services of a resource express the dependencies between resource classes. The arrows between the resources represent the services required and provided. Planware analyzes the task and resource models to determine a hierarchy of service matches (service required matched with service offered) that is rooted in a task model.

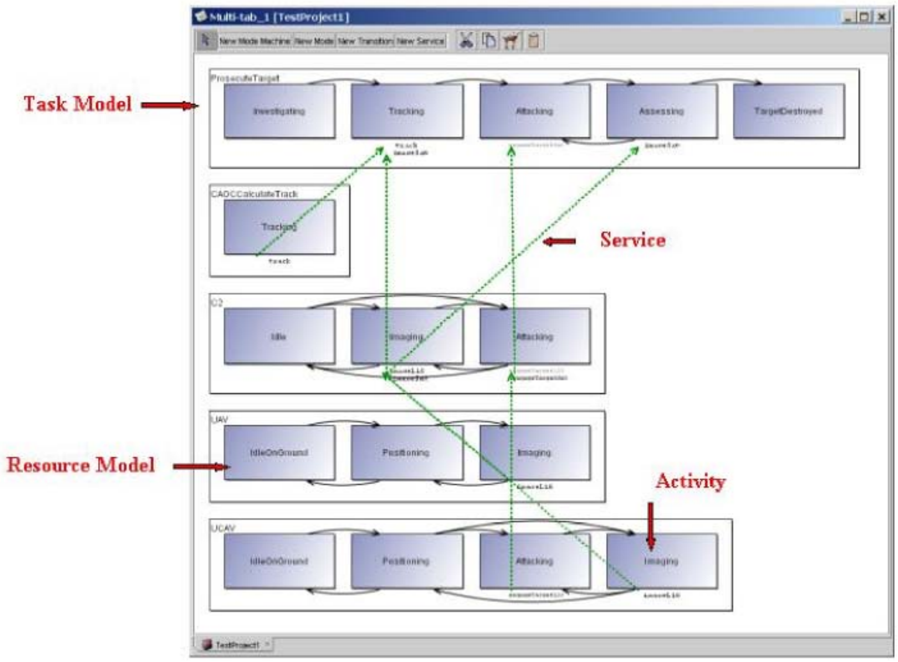

Figure 19:Planware Model for Sensor to Decision Maker to Shooter Scheduler

Algorithm Design – The problem specification is used to automatically instantiate program schemes that embody abstract algorithmic knowledge about global search and constraint propagation. The algorithm generation process follows the structure of the service hierarchy, resulting in a nested structure of instantiated search schemes.

Datatype Refinement and Optimization – Program schemes used by the generator are described in terms of abstract datatypes. After the generation of the scheduling algorithm, abstract datatypes are refined to concrete programming language types. Additional program transformations to provide further optimizations can then be manually or automatically applied to the resulting code.

Code generation – Finally code in a programming language (currently CommonLisp) is generated.

7. Conclusion: Summary & Related Work

In its “e-Merge-ANT” project, Kestrel developed notions of real-time, approximate optimization of constraint networks and developed a distributed algorithm that is simple but appears to be effective for a wide range of problems. It seems that prior to this work, constraint problems were generally considered too computationally expensive to solve or to be considered for real-time applications.

The idea of local constraint optimization that is the keystone of the algorithm presented in this report is not a new one. Indeed, a similar algorithm was published by Fabiunke [3]. However, what is new is Kestrel’s focus on the real-time properties, and in particular on how the algorithm behaves after a small number of iterations (in the spirit of “good-enough, soon-enough”). Kestrel tailored the algorithm mainly to improve this part of its performance.

Other research groups are investigating distributed constraint satisfaction and optimization. For example, Yokoo et al. have developed what is perhaps the best known algorithm in the field, the “Distributed Breakout” algorithm. Investigations by Zhang et al. under the ANTs program have generally found that the Kestrel algorithm gives better performance [5]. Kestrel’s algorithm also served as a subject of study as regards phase transitions by van Parunak et al. [1,2].

Kestrel used the framework of distributed constraint optimization to develop a solution for distributed resource management in sensor networks. The solution concept seems validated by the results on the challenge problem simulator, although the results on the hardware indicate that details for a practical system would need to be worked out. This approach to distributed resource management is being taken up by others [4].

In its work on time-critical targeting, Kestrel was able to model at a high level the essential properties of air resources and targets and to automatically generate executable schedulers. Due to security restrictions, Kestrel’s models had to be based on publicly available information [6,7,8,9] and thus are inevitably somewhat simplified and naïve. Nevertheless, Kestrel believes that the basic approach has been validated.

Publications

Distributed Coordination through Anarchic Optimization, Stephen Fitzpatrick & Lambert Meertens, Chapter in Distributed Sensor Networks: A Multiagent Perspective, pp. 257-295; Victor Lesser, Charles L. Ortiz, Jr., & Milind Tambe (Eds.); Kluwer Academic Publishers, 2003; ISBN: 1-4020-7499-9

Experiments on Dense Graphs with a Stochastic, Peer-to-Peer Colorer, Stephen Fitzpatrick & Lambert Meertens, Probabilistic Approaches in Search, Workshop at Eighteenth National Conference on Artificial Intelligence (AAAI 2002), 28 July 2002, Edmonton, Alberta, Canada, Carla Gomes & Toby Walsh (Ed.), AAAI Press, Technical Report WS-02-14, ISBN 1-57735-167-3, pp. 24-28

Asynchronous Execution and Communication Latency in Distributed Constraint Optimization, Lambert Meertens & Stephen Fitzpatrick, Proceedings of The Third International Workshop on Distributed Constraint Reasoning, First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002), 16 July 2002, Bologna, Italy, Makoto Yokoo (Ed.), pp. 80-85

Scalable, Anytime Constraint Optimization through Iterated, Peer-to-Peer Interaction in Sparsely-Connected Networks, Stephen Fitzpatrick & Lambert Meertens, Proceedings of The Sixth Biennial World Conference on Integrated Design & Process Technology (IDPT 2002), 23-28 June 2002, Pasadena, California, U.S.A., H. Ehrig, B.J. Kramer & A. Ertas (Ed.), Society for Design and Process Science, ISSN No. 1090-9389

Peer-to-Peer Coordination of Autonomous Sensors in High-Latency Networks using Distributed Scheduling and Data Fusion, Lambert Meertens & Stephen Fitzpatrick, Technical Report KES.U.01.09, December 2001, Kestrel Institute, Palo Alto, California

An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs, Stephen Fitzpatrick & Lambert Meertens, 1st Symposium on Stochastic Algorithms: Foundations and Applications, 13-14 December 2001 Berlin, Germany, Lecture Notes in Computer Science 2264, Kathleen Steinhoefel (Ed.), Springer-Verlag, ISBN 3-540-43025-3, pp. 49-64

Soft, Real-Time, Distributed Graph Coloring using Decentralized, Synchronous, Stochastic, Iterative-Repair, Anytime Algorithms - A Framework, Stephen Fitzpatrick & Lambert Meertens, Technical Report KES.U.01.05, May 2001, Kestrel Institute, Palo Alto, California

References

[1] *How to Calm Hyperactive Agents,* H. Van Dyke Parunak, Sven Brueckner, Robert Matthews, & John Sauter, The Second International Joint Conference on Autonomous Agents & Multiagent Systems, ACM 2003, ISBN 1-58113-683-8, pp. 1092-1093

[2] *Information-driven Phase Changes in Multi-agent Coordination*, Sven Brueckner & H. Van Dyke Parunak, The Second International Joint Conference on Autonomous Agents & Multiagent Systems, ACM 2003, ISBN 1-58113-683-8, pp. 950-951

[3] *Parallel Distributed Constraint Satisfaction*, M. Fabiunke, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications 1999 (PDPTA'99), pp. 1585–1591..

[4] *Self-Tuning Energy-Aware Multichannel (STEAM) Scheduling*, Umesh Shankar, Tech. Report UCB//04-1300 University of California, Berkeley, March 2004

[5] *A Comparative Study of Distributed Constraint Algorithms*, Weixong Zhang, Guandong Wang, Zhao Xing & Lars Wittenberg, chapter in Distributed Sensor Networks, Victor Lesser, Charles L. Ortiz, Jr., & Milind Tambe (eds.), Kluwer Academic Publishers, 2003, ISBN: 1-4020-7499-9, pp. 319-338

[6]*Federation of American Scientists: Military Analysis Network*, http://www.fas.org/man/index.html

[7] *Air Force Technology: Industry Projects*, http://www.airforce-technology.com/projects/index.html

[8] *Army Technology: Industry Projects*, http://www.army-technology.com/projects/index.html

[9] *USAF Intelligence Targeting Guide Pamphlet 14- 210 Intelligence*, http://www.fas.org/irp/doddir/usaf/afpam14-210/index.html